

## 13.1 イテレータメソッド (Iterator Method)

イテレータは、next と hasNext のほかにも豊富なメソッドをサポートしており、それを次で説明します。それらメソッドの多くは、リスト機能の対応するものに似ています。

**Append** メソッド append は、新しいイテレータを構築します。構築されるイテレータは、元のイテレータの最後まで到達すると、与えられたイテレータで続きます。

```
def append[B >: A](that: Iterator[B]): Iterator[B] = new Iterator[B] {
  def hasNext = Iterator.this.hasNext || that.hasNext
  def next = if (Iterator.this.hasNext) Iterator.this.next else that.next
}
```

append 定義にある Iterator.this.next という項と Iterator.this.hasNext という項は、それを取り囲む Iterator クラスで定義されている、対応するメソッドを呼び出します。もし、this に対するプレフィックス Iterator がなければ、hasNext と next は、append の結果 (オブジェクト) に定義されているメソッド自身を再帰的に呼び出してしまいます。これは我々の望むことではありません。

**Map, FlatMap, Foreach** メソッド map は、元のイテレータのすべての要素を、与えられた関数 f で変換して返すイテレータを構築します。

```
def map[B](f: A => B): Iterator[B] = new Iterator[B] {
  def hasNext = Iterator.this.hasNext
  def next = f(Iterator.this.next)
}
```

メソッド flatMap はメソッド map と似ていますが、変換する関数 f がイテレータを返す点が違います。flatMap の結果は、f を順に呼んでいって返されてくるイテレータ達を結合したものです。

```
def flatMap[B](f: A => Iterator[B]): Iterator[B] = new Iterator[B] {
  private var cur: Iterator[B] = Iterator.empty
  def hasNext: Boolean =
    if (cur.hasNext) true
    else if (Iterator.this.hasNext) { cur = f(Iterator.this.next); hasNext }
    else false
  def next: B =
    if (cur.hasNext) cur.next
    else if (Iterator.this.hasNext) { cur = f(Iterator.this.next); next }
    else error("next on empty iterator")
}
```

map に深く関係するのが foreach メソッドです。与えられた関数をイテレータのすべての要素に適用しますが、結果のリストを構築しません。

```
def foreach(f: A => Unit): Unit =
  while (hasNext) { f(next) }
```

**Filter** メソッド filter は、元のイテレータのすべての要素のうち、基準 p を満たすものを返すイテレータを構築します。

```
def filter(p: A => Boolean) = new BufferedIterator[A] {
  private val source =
    Iterator.this.buffered
  private def skip
    { while (source.hasNext && !p(source.head)) { source.next } }
  def hasNext: Boolean =
    { skip; source.hasNext }
  def next: A =
    { skip; source.next }
  def head: A =
    { skip; source.head }
}
```

filter はイテレータの「バッファされた」サブクラスのインスタンスを返します。BufferedIterator オブジェクトは、そのほかに head メソッドを持っています。このメソッドは、head (訳注: next の誤り?) が返すはずの要素を返しますが、next のように、要素をそれ以降に進めることはありません。そのため、head が返す要素は、次の head または next の呼び出しで再び返されます。BufferedIterator トレイトの定義は次のとおりです。

```
trait BufferedIterator[+A] extends Iterator[A] {  
  def head: A  
}
```

map, flatmap, filter, foreach がイテレータに存在するので、for 内包表記 と for ループ がイテレータに対しても使えます。たとえば、1 から 100 までの数の平方を表示する適用は、次のように等価に表現できます。

```
for (i < Iterator.range(1, 100))  
  println(i * i)
```

Zip メソッド zip は、他のイテレータをとって、2 つのイテレータから返される要素のペアからなるイテレータを返します。

```
def zip[B](that: Iterator[B]) = new Iterator[(A, B)] {  
  def hasNext = Iterator.this.hasNext && that.hasNext  
  def next = {Iterator.this.next, that.next}  
}
```

[前ページ](#) [13章](#) [目次](#) [次ページ](#)

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