

13.2 イテレータの構築 (Constructing Iterators)

具体的なイテレータは、クラス `Iterator` の2つの抽象メソッド `next` と `hasNext` の実装を提供する必要があります。もっとも単純なイテレータは、常に空の列を返す `Iterator.empty` です。

```
object Iterator {
  object empty extends Iterator[Nothing] {
    def hasNext = false
    def next = error("next on empty iterator")
  }
}
```

もっと面白みのあるイテレータは、配列のすべての要素を列挙するものです。このイテレータはオブジェクト `Iterator` で定義されている `fromArray` メソッドで構築されます。

```
def fromArray[A](xs: Array[A]) = new Iterator[A] {
  private var i = 0
  def hasNext: Boolean =
    i < xs.length
  def next: A =
    if (i < xs.length) { val x = xs(i); i += 1; x }
    else error("next on empty iterator")
}
```

ほかのイテレータとして、範囲内の整数を列挙するものがあります。 `Iterator.range` 関数は、与えられた範囲の整数値をたどるイテレータを返します。

```
object Iterator {
  def range(start: Int, end: Int) = new Iterator[Int] {
    private var current = start
    def hasNext = current < end
    def next = {
      val r = current
      if (current < end) current += 1
      else error("end of iterator")
      r
    }
  }
}
```

ここまで見てきたイテレータはいずれ終わりますが、永遠に続くイテレータも定義できます。たとえば、次のイテレータは初期値からずっと続く整数を返します(*1)。

```
def from(start: Int) = new Iterator[Int] {
  private var last = start + 1
  def hasNext = true
  def next = { last += 1; last }
}
```

(*1) `int` 型が有限の表現であるため、 2^{31} で数は元に戻ります。

[前ページ](#) [13章 目次](#) [次ページ](#)

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

投稿