rayListIterator 的简化。

```
1  public class MyArrayList<AnyType> implements Iterable<AnyType>
2  {
3      private int theSize;
4      private AnyType [ ] theItems;
5          ...
6      public java.util.Iterator<AnyType> iterator( )
7        { return new ArrayListIterator<AnyType>( this ); }
8
9      private static class ArrayListIterator<AnyType>
10                                          implements java.util.Iterator<AnyType>
11     {
12         private int current = 0;
13         private MyArrayList<AnyType> theList;
14               ...
15         public ArrayListIterator( MyArrayList<AnyType> list )
16           { theList = list; }
17
18         public boolean hasNext( )
19           { return current < theList.size( ); }
20         public AnyType next( )
21           { return theList.theItems[ current++ ]; }
22     }
23 }
```

图 3-19　迭代器 3 号版本(能够使用)：迭代器是一个嵌套类并存储当前位置和一个连接到 MyArrayList 的链。它能够使用是因为该嵌套类被认为是 MyArrayList 类的一部分

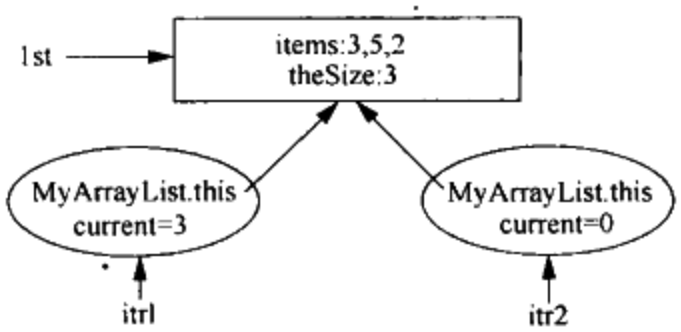

图 3-20　带有内部类的迭代器/容器

首先，ArrayListIterator 是隐式的泛型类，因为它现在依赖于 MyArrayList，而后者是泛型的；我们可以不必说这些。

其次，theList 没有了，我们用 size()和 theItems[current++]作为 MyArrayList.this.size()和 MyArrayList.this.theItems[current++]的简记符。theList 作为数据成员，它的去除也删除了相关的构造器，因此程序又转变成 1 号版本的样式。

我们可以通过调用 MyArrayList 的 remove 来实现迭代器的 remove 方法。由于迭代器的 remove 可能与 MyArrayList 的 remove 冲突，因此我们必须使用 MyArrayList. this. remove。注意，在该项被删除之后，一些元素需要移动，因此 current 被视为同一元素也必须移动。于是，我们使用 -- 而不是 - 1。

内部类为 Java 程序员带来句法上的便利。它们不需要编写任何 Java 代码，但是它们在语言中的出现使 Java 程序员以自然的方式(如 1 号版本那样)编写程序，而编译器则编写使内部类对象和外部类对象相关联所需要的附加代码。