实现时要大。findKth操作不如数组实现时的效率高；findKth(i)花费 $O(i)$的时间并以这种明显的方式遍历链表而完成。在实践中这个界是保守的，因为调用 findKth 常常是以(按 i)排序后的方式进行。例如，findKth(2),findKth(3),findKth(4)以及 findKth(6)可通过对表的一次扫描同时实现。

remove 方法可以通过修改一个 next 引用来实现。图 3-2 给出在原表中删除第三个元素的结果。

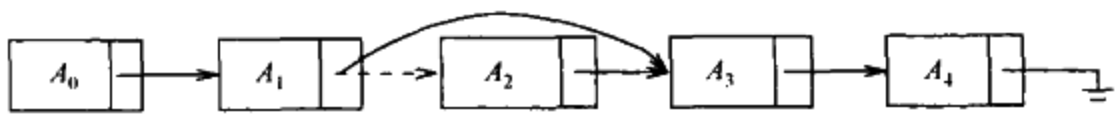

图 3-2 从链表中删除

insert 方法需要使用 new 操作符从系统取得一个新节点，此后执行两次引用的调整。其一般想法在图 3-3 中给出，其中的虚线表示原来的 next 引用。

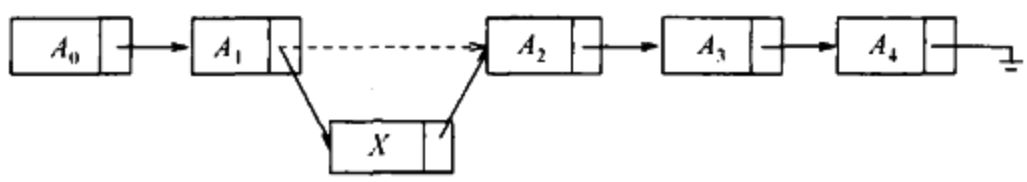

图 3-3 向链表插入

我们看到，在实践中如果知道变动将要发生的地方，那么向链表插入或从链表中删除一项的操作不需要移动很多的项，而只涉及常数个节点链的改变。

在表的前端添加项或删除第一项的特殊情形此时也属于常数时间的操作，当然要假设到链表前端的链是存在的。只要我们拥有到链表最后节点的链，那么在链表末尾进行添加操作的特殊情形(即让新的项成为最后一项)可以花费常数时间。因此，典型的链表拥有到该表两端的链。删除最后一项比较复杂，因为必须找出指向最后节点的项，把它的 next 链改成 null，然后再更新持有最后节点的链。在经典的链表中，每个节点均存储到其下一节点的链，而拥有指向最后节点的链并不提供最后节点的前驱节点的任何信息。

保留指向最后节点的节点的第 3 个链的想法行不通，因为它在删除操作期间也需要更新。我们的做法是，让每一个节点持有一个指向它在表中的前驱节点的链，如图 3-4 所示，我们称之为**双链表**(doubly linked list)。

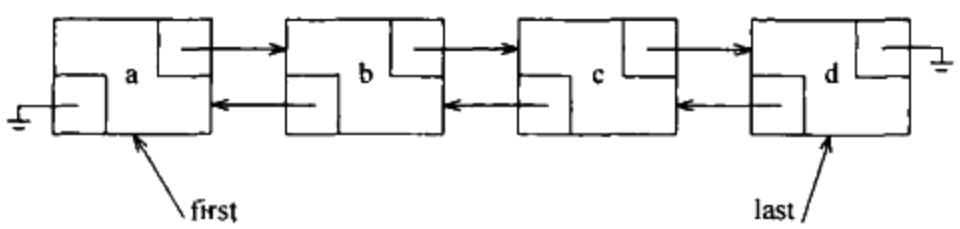

图 3-4 双链表

3.3 Java Collections API 中的表

在类库中，Java 语言包含有一些普通数据结构的实现。该语言的这一部分通常叫做 **Collections API**。表 ADT 是在 Collections API 中实现的数据结构之一。我们将在第 4 章看到其他一些数据结构。

3.3.1 Collection 接口

Collections API 位于 java.util 包中。集合(collection)的概念在 Collection 接口中得到抽象，它存储一组类型相同的对象。图 3-5 显示该接口一些最重要的部分(但一些方法未被显示)。