**Iterator（迭代器）**

意图

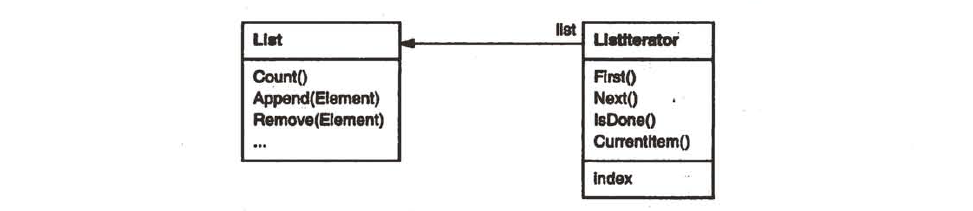
提供一种方法顺序访问一个聚合对象中各个元素，而又不暴露该对象的内部表示。

动机

一个聚合对象，如列表（list），应该提供一种方法来让别人可以访问它的元素，又不需暴露它的内部结构，此外，针对不同的需要，可能要以不同的方式遍历这个列表。但是即使可以预见到所需的那些遍历操作，你可能也不希望列表的接口中充斥着各种不同遍历的操作。有时还可能需要在同一列表上同时进行多次遍历。

迭代器模式可以帮你解决所有的这些问题。这一模式的关键思想是将对列表的访问和遍历从列表对象中分离出来并放入一个迭代器对象中。迭代器类定义了一个访问该列表元素的接口。迭代器对象负责跟踪当前的元素；即，它知道哪些元素已经遍历过了。

例如，一个列表类可能需要一个列表迭代器，它们之间的关系如下图：



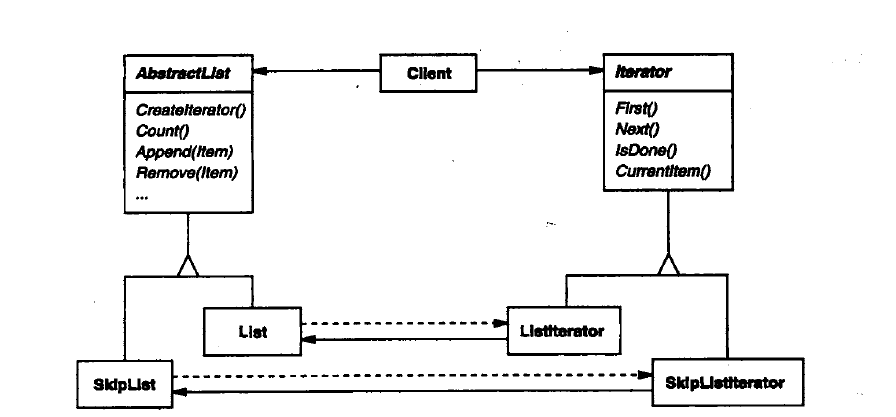
在实例化列表迭代器之前，必须提供待遍历的列表，一旦有了该列表迭代器的实例，就可以顺序地访问该列表的各个元素。CurrentItem操作返回列表中的当前元素，First操作初始化迭代器，使当前元素指向列表的第一个元素，Next操作将当前元素指针向前推进一步，指向下一个元素，而IsDone检查是否已越过最后一个元素，也就是完成了这次遍历。

将遍历机制在与对象列表分离使我们可以定义不同的迭代器来实现不同的遍历策略，而无需在列表接口中列举它们。例如，过滤表列迭代器可能只访问那些满足特定过滤约束条件的元素。

注意迭代器和列表是耦合在一起的，而且客户对象必须知道遍历的是一个列表而不是其他聚合结构。最好能有一种方法使得不需要改变客户代码即可改变该聚合类。可以通过将迭代器的概念推广到多态迭代来达到这个目标。

例如，假定我们还有一个列表的特殊实现。比如说SkipList。SkipList是一种具有类似于平衡树性质的随机数据结构。我们希望我们的代码对List和SkipList对象都适用。

首先，定义一个抽象列表类AbstractList，它提供操作列表的公共接口。类似的，我们也需要一个抽象迭代器类Iterator，它定义公共的迭代接口。然后我们可以为每个不同的列表实现定义具体的Iterator子类。这样迭代机制就与具体聚合类无关了。



余下的问题是如何创建迭代器。既然要使这些代码不依赖于具体的列表子类，就不能仅仅简单地实例化一个特定的类，而要让列表对象负责创建相应的迭代器。这需要列表对象提供CreateIterator这样的操作，客户请求调用该操作以获得一个迭代器对象。

创建迭代器是一个Factory Method模式例子。我们在这里用它来使得一个客户可向一个列表对象请求合适的迭代器。Factory Method模式产生两个类层次，一个是里诶报的，一个是迭代器的。CreateIterator“联系”两个类层次。

结构图

