List 接口

An ordered collection (also known as a *sequence*). The user of this interface has precise control

over where in the list each element is inserted. The user can access elements by their integer index (position in the list), and search for elements in the list.

Unlike sets, lists typically allow duplicate elements. More formally, lists typically allow pairs of elements e1 and e2 such that e1.equals(e2), and they typically allow multiple null elements if they allow null elements at all. It is not inconceivable that someone might wish to implement a list that prohibits duplicates, by throwing runtime exceptions when the user attempts to insert them, but we expect this usage to be rare.

一个有序的集合(以队列而知名)。使用这个接口可以精确控制每个被插入list中的元素。用户可以使用它们的下标访问元素和在列表list检索元素。

不像set集合，list典型的允许重复元素。更正式些，list代表允许一对像e1=e2的元素。并且它们也允许多个null元素。一些人也许希望实现一个禁止重复元素的list，当用户尝试插入重复数据时，抛出运行时异常，这是很平常的，但是我们期望这种习惯很稀少。

The List interface places additional stipulations, beyond those specified in the Collection interface, on the contracts of the iterator, add, remove, equals, and hashCode methods. Declarations for other inherited methods are also included here for convenience.

The List interface provides four methods for positional (indexed) access to list elements. Lists (like Java arrays) are zero based. Note that these operations may execute in time proportional to the index value for some implementations (the LinkedList class, for example). Thus, iterating over the elements in a list is typically preferable to indexing through it if the caller does not know the implementation.

The List interface provides a special iterator, called a ListIterator, that allows element insertion and replacement, and bidirectional access in addition to the normal operations that the Iterator interface provides. A method is provided to obtain a list iterator that starts at a specified position in the list.

The List interface provides two methods to search for a specified object. From a performance standpoint, these methods should be used with caution. In many implementations they will perform costly linear searches.

The List interface provides two methods to efficiently insert and remove multiple elements at an arbitrary point in the list.

Note: While it is permissible for lists to contain themselves as elements, extreme caution is advised: the equals and hashCode methods are no longer well defined on such a list.

Some list implementations have restrictions on the elements that they may contain. For example, some implementations prohibit null elements, and some have restrictions on the types of their elements. Attempting to add an ineligible element throws an unchecked exception, typically NullPointerException or ClassCastException. Attempting to query the presence of an ineligible element may throw an exception, or it may simply return false; some implementations will exhibit the former behavior and some will exhibit the latter. More generally, attempting an operation on an ineligible element whose completion would not result in the insertion of an ineligible element into the list may throw an exception or it may succeed, at the option of the implementation. Such exceptions are marked as "optional" in the specification for this interface.