特性和使用场景：

List：

a.是 Collection 接口的子接口。

b.元素有序，即输出顺序与输入顺序是一致的，可以利用索引值进行操作。

c.允许其中元素重复。

d.存放单列元素。

（1）Arraylist

a.底层是可变数组：Objtce[] ElmentDate。

b.线程不安全。

c.可以存在任意类型的引用类型的数据，包括 null，可以存放多个 null。

d.更加适用于查询较多的情况，由于 java 程序中大多数业务均为查询业务，因此，使用广泛。

（2）Linkedlist

a.底层是双向链表。

b.适用于增删较多的情形。

（3）Vectory

a.与 Arraylist 使用相似。

b.线程安全，但是相对于 Arraylist 效率较低。

Set:

a.元素无序，不能利用索引值进行相应的操作。

b.元素不能重复，即最多只能存放一个 null 值。

c.存放单列元素。

（1）Hashset

a.底层使用了 Hashmap。

b.底层使用的数据结构是数组 hashmap$node[] table、链表与红黑树。

c.数据的去重规则：首先利用其 hash 值确定数据在 table 表中的索引值，该索引处没有存放数据，可以直接插入，若该索引处已经存放数据，则利用 equals 方法，逐一进行比较，如果有相同的元素，不加入，没有相同的元素，将在链表的最后方加入。

d.适用于无序且不允许重复的情况。

（2）linkedhashset

a.底层使用了 linkedhashmap。

b.底层使用的是数组 hashmap$node[] table 和双向链表。

c.其中链表之中存放的数据类型是：linkedhashmap$entry 类型。

d.由于底层使用的是双向链表可以保证插入顺序与取出顺序的一致。

（2）treeset

a.有顺序。

b.底层使用了 treemap。

c.数据的去重规则：利用传入的 comparator 接口的 compare 方法进行去重。如果没有重写 copare 方法，则利用 comparable 接口的 compareTo 方法进行去重，所有插入对象的类要实现 comparable 接口。

d.适用于存放单列元素有序的情况。

Map

a.存放双列元素 key-value。

b.key 不可以重复，因此只能存放一个 null 值

c.value 可以重复，可以存放多个 null 值。

d.key 与 value 之间存在映射关系，两者一一对应，可以通过 key 查找 value。

e.key 与 value 的数据类型均为 Object。

（1） Hashmap

a.适用于存放双列元素，无序的情况。

b.key 相同时，插入元素，value 会被替换。

c.key 的去重规则与 hashset 相同。

d.线程不安全。

（2）Treemap

a.适用于存放双列元素且有序的情况，

b.利用 comparator 接口的 compare 方法进行去重（与 treeset 相同）

（3）Hashtable

a.线程安全。

b.不允许存放 null，会抛出异常

c.拥有子类 properties：可以读取 xx.properties 文件之中的数据并加载到 properties 类对象之中，进行读取与修改。