1. 集合

Collection接口:

List接口:(实现类都是有序、可重复，并且都实现了Cloneable和Serializable)

ArrayList:

底层是基于动态数组实现的，能存放null元素，线程不安全，但是可以使 用Collections.synchrionizedList(new ArrayList<>())达到线程安全。他的初始 大小为10，每次扩充增加原来的0.5倍(只取整数部分)。

随机查询单个元素快，插入、删除元素速度慢。

删除元素时不会减少容量，可以使用调用trimToSize()来减少容量。

LinkedList:

底层是基于双向循环链表实现的，可以存放null元素，线程不安全，但是 可以使用Collections.synchrionizedList(new ArrayList<>())达到线程安全。没 有初始化大小，也没有扩容机制，可以一直插入新的与元素。

随机查询单个元素慢，因为要根据指针位移，插入、删除元素快。

Vector:

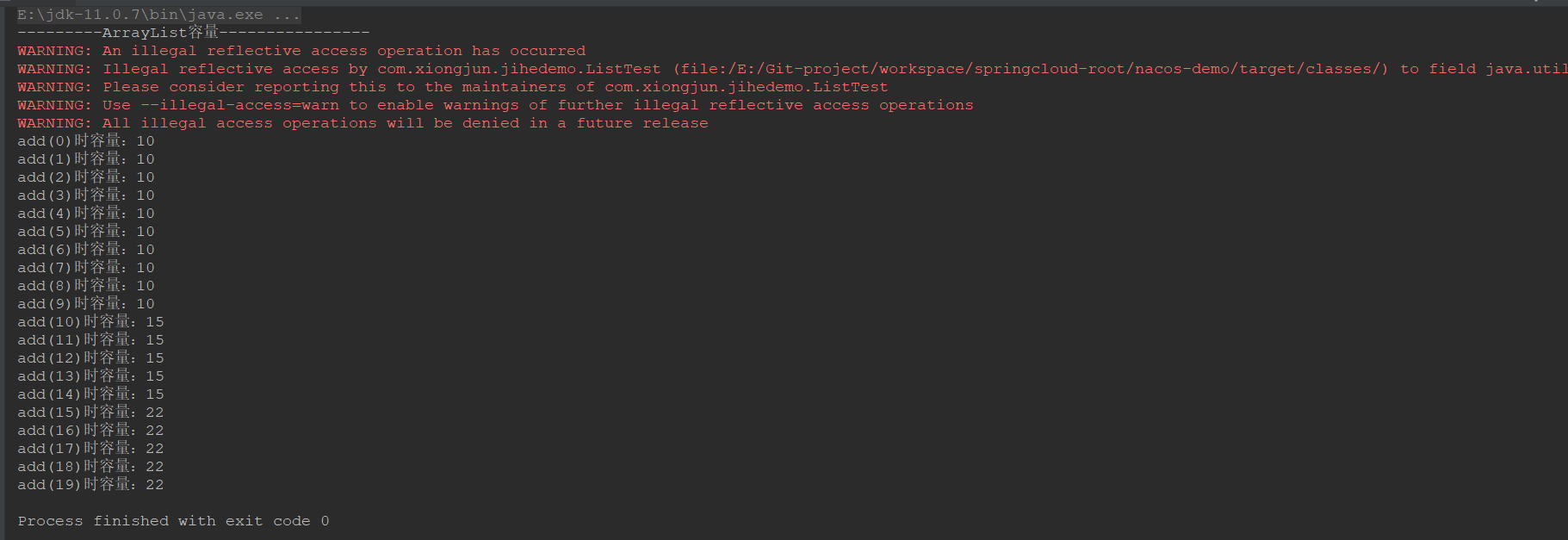
底层也是基于动态数据实现的，能存放null元素，线程安全，初始化容量也是10，每次扩容增加原来的一倍。

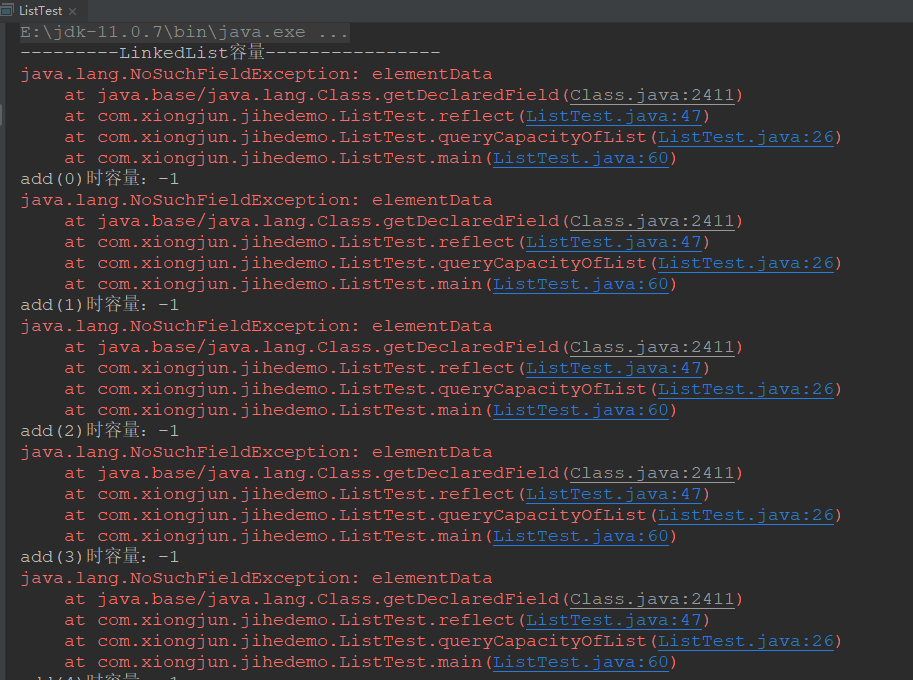
如下图，通过代码来验证扩容的机制：

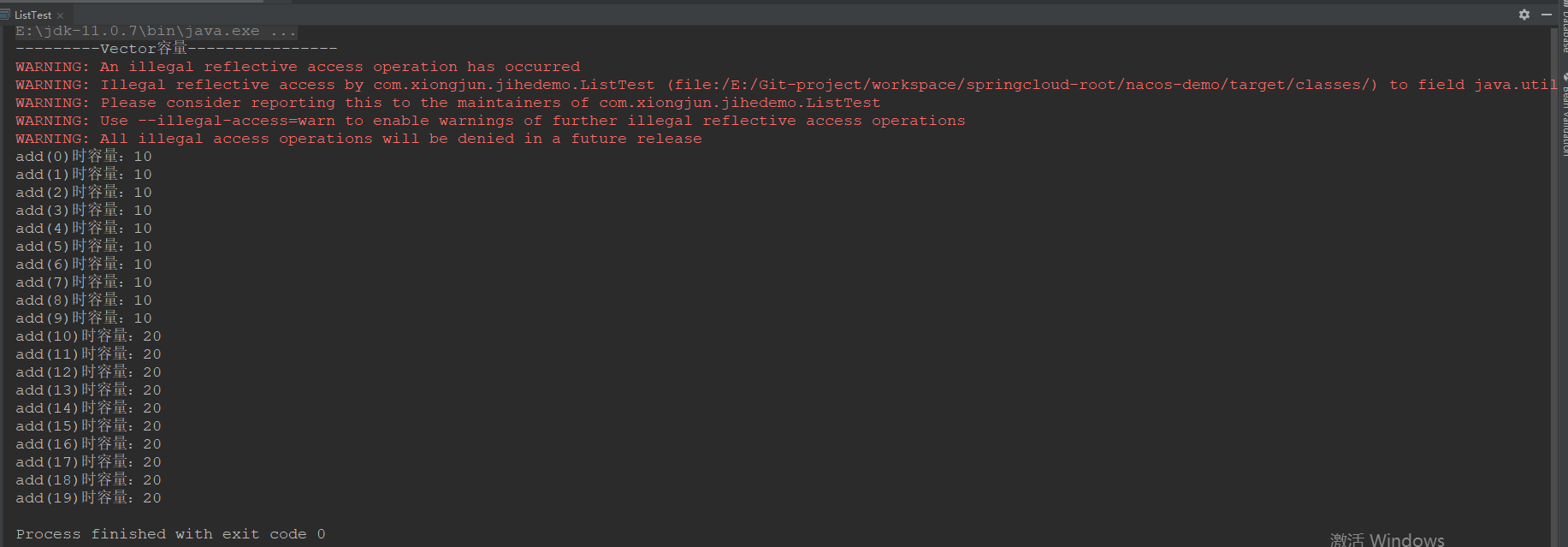
代码：



执行结果：







Set接口(实现类中的元素都是不可重复的，并且都实现了Cloneable和Serializable)):

HashSet:

元素不可重复、无序,但是元素在set中的位置是有该元素的HashCode决定的，其位置其实是固定的。元素可以为null。

底层数据结构是哈希表。数据存储在HashMap中。HashSet进行构造时，

除了可以使用Collection进行构造外，基本都调用了HashMap的构造函数完成。 所以初始化容量为16，加载因子是0.75。线程不安全。

LinkedHashSet:

元素不可重复、有序。元素可以为null。

底层数据结构是链表+hash表，由链表保证元素有序、hash表保证元素唯一。LinkedHashSet使用LinkedHashMap对象来存储它的元素。线程不安全。

当遍历该集合时候，LinkedHashSet将会以元素的添加顺序访问集合的元素。LinkedHashSet在迭代访问Set中的全部元素时，性能比HashSet好，因为它是根据链表维护元素的次序的，但是插入时性能稍微逊色于HashSet(百万数据量时才能体现出)

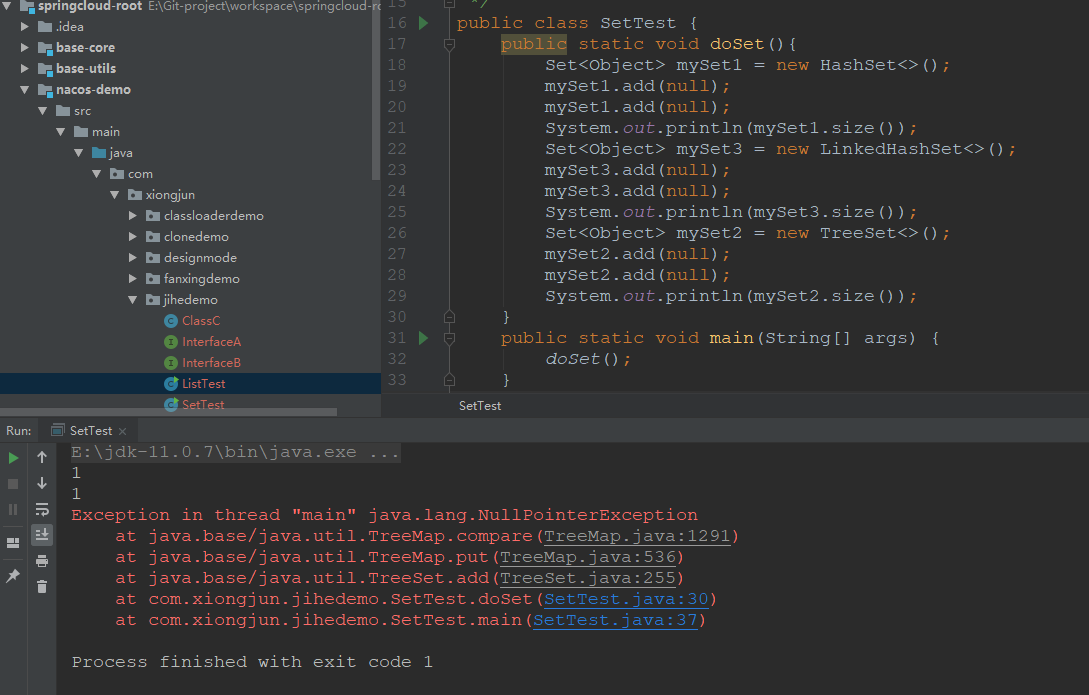
TressSet:

元素不可重复、有序，根据比较的返回值是否是0来决定是否唯一。

底层数据结构是红黑树。

元素不能为null，否则会报空指针异常。它使用了TreeMap来控制数据的有序性。线程不安全。

TreeSet添加元素时，只能添加同类型的元素，否则会报类型转换错误。



TreeSet有以下两种排序方式：

自然排序:实现Comparable接口，重写了compareTo()来进行排序。

自定义排序(比较器排序):实现Comparable接口或者Comparator接口，重写compareTo()方法。两种排序方式对比如下：

TreeSet-自然排序：





执行结果：



TreeSet-自定义排序：







执行结果：



Queue接口:

都是先进先出的顺序。

阻塞队列与非阻塞队列区别：

当队列是空时，从队列中获取元素的操作会被阻塞，直到其他线程往空的队列插入新的元素；当队列满时，往队列插入新的元素同样会被阻塞，直到队列有位置可以插入。

1.ArrayDeque, （数组双端队列，非阻塞）

2.PriorityQueue, （优先级队列，有界，非阻塞）

3.ConcurrentLinkedQueue, （基于链表的并发队列，非阻塞）

4.DelayQueue, （延期阻塞队列，误解）（阻塞队列实现了BlockingQueue接口）

5.ArrayBlockingQueue, （基于数组的并发阻塞队列）

6.LinkedBlockingQueue, （基于链表的FIFO阻塞队列）

7.LinkedBlockingDeque, （基于链表的FIFO双端阻塞队列）

8.PriorityBlockingQueue, （带优先级的无界阻塞队列）

9.SynchronousQueue （并发同步阻塞队列）

AbstractQueue抽象类：

PriorityQueue(优先队列)：

Map接口：

HashMap:

TreeMap:

HashTable: