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:

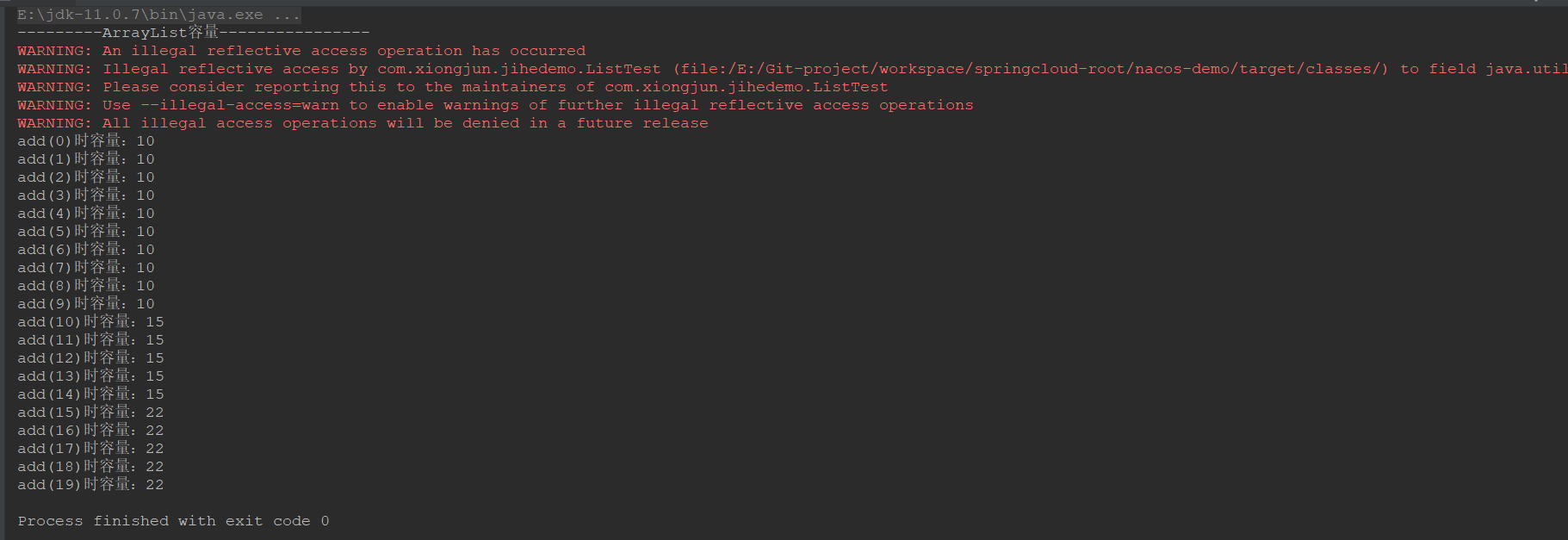
底层也是基于动态数据实现的，能存放null元素，线程安全，初始化容量也是10，每次扩容增加原来的一倍。

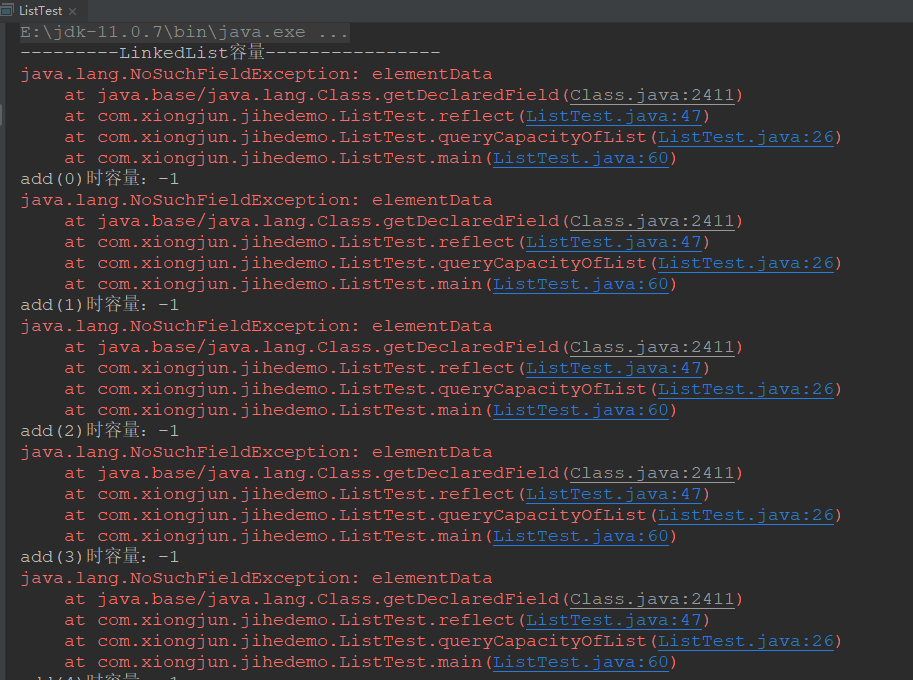
如下图，通过代码来验证扩容的机制：

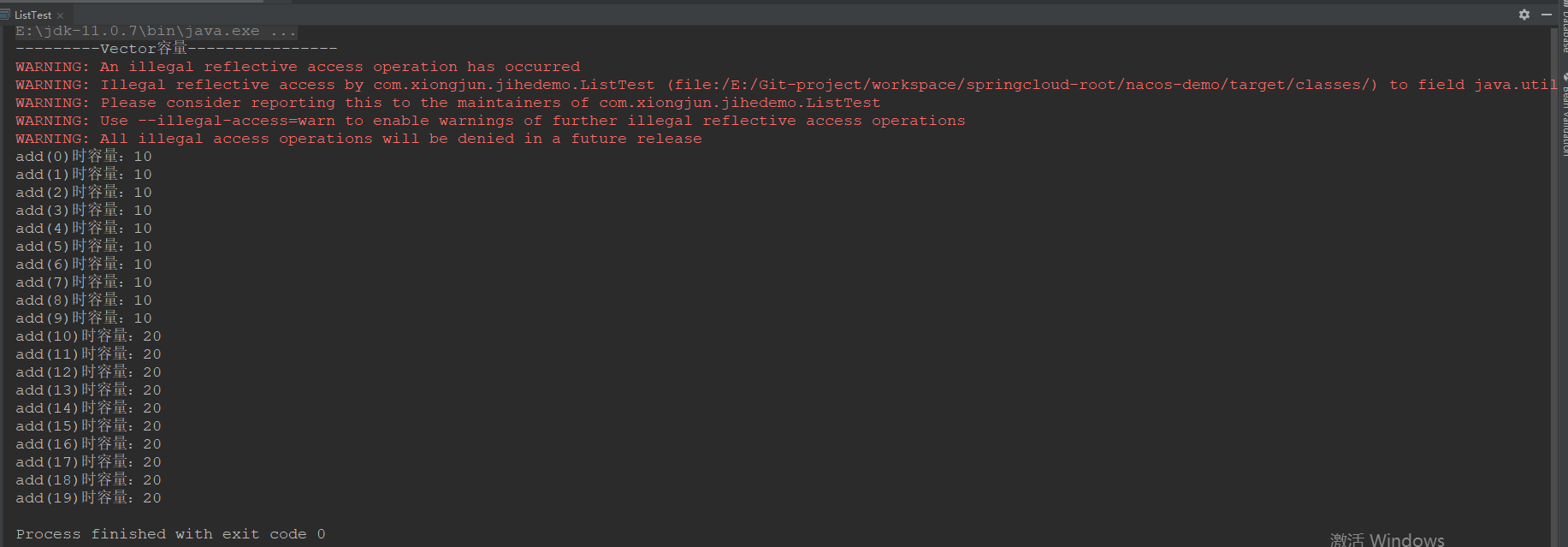
代码：



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)

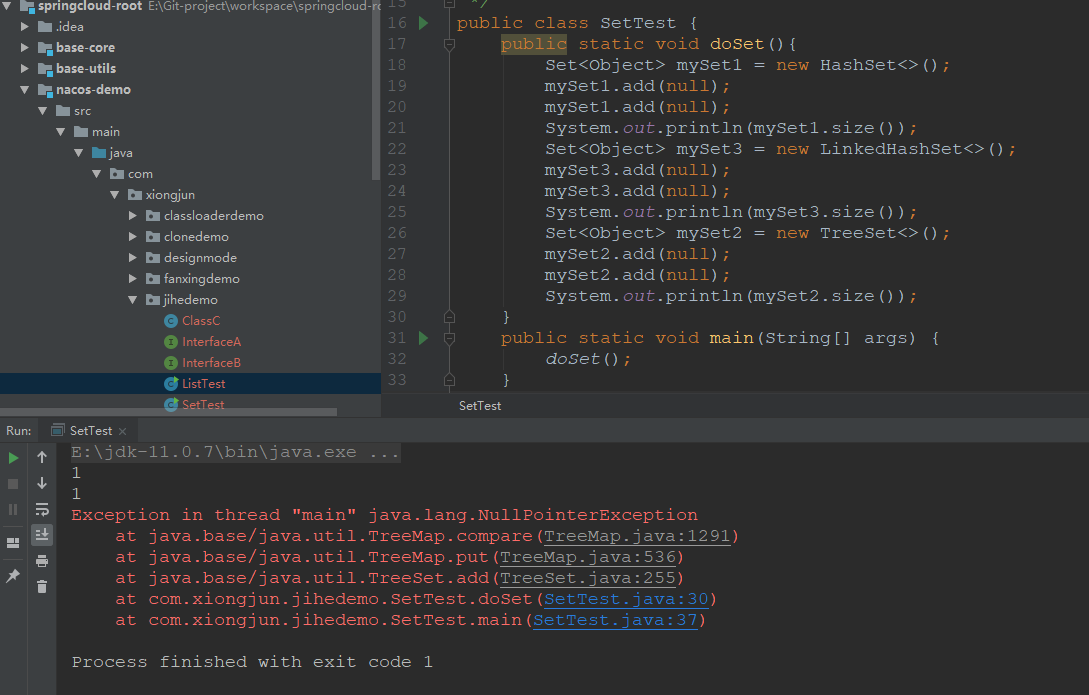
TressSet:

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）

Map接口：

HashMap: 无序，但是是按照一定的顺序(key的hash值顺序)排列;key唯一且可 以为null，value可以为null;数据结构是数组+链表,当多个key的hash值 相等时，会形成链表存储，所以数组里的每个元素都是一个单向链表;默 认的容量大小为16，加载因子为0.75,扩容后容量增加一倍; 非线程安 全;jdk1.8之后，数据结构改为数组+链表+红黑树，当数组中的链表元素长 度>8且数组容量达到64个时，会由链表转换为红黑树，而当红黑树的节 点<6时，会由红黑树转换为链表；非线程安全;

为什么是8？

如果 hashCode 分布良好，也就是 hash 计算的结果离散好的话，那么红黑树这种形式是很少会被用到的，因为各个值都均匀分布，很少出现链表很长的情况。在理想情况下，链表长度符合泊松分布，各个长度的命中概率依次递减，当长度为 8 的时候，概率仅为 0.00000006。这是一个小于千万分之一的概率。

HashMap的读取过程与存储过程类似，先获取key的hash值在table中对应的位置，然后遍历该链获取相同的key的值即可。

TreeMap: 有序，分为默认排序和自定义排序;key唯一，且不能为null必须要实现 Comparable接口，从写compareTo()方法，或者实现Comparator接口，重 写compare()方法和equals()方法，value可以为null;数据结构是红黑树;非 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; 非线程安全;

LinkedHashMap: 有序，按插入的先后顺序排列，key唯一且可以为null，value可 以为null;因为它继承了HashMap,所以它的默认容量也是16，扩容因子也 是0.75，扩容后容量增加一倍；非线程安全；

LinkedHashMap底层就是数组+双向链表，使用双向链表保证了节点的有序性。

Hashtable: 无序;key唯一，key和value都不能为空; 线程安全；数据结构与 HashMap相似，数组+单向链表。默认容量是11(因为除质数求余的分散 效果好)，扩容因子是0.75，扩容后的元素的容量是扩容前容量的2倍 + 1；

Hashtable的sychronized是针对整个表的。

ConcurrentHashMap: 无序;key唯一，key和value都不能为空; 线程安全；底层采 用分段(Segment)数组+链表,Segment本身就相当于 一个 HashMap对象， 相当于是二级数组因为是分段的，所以，ConcurrentHashMap允许多个修 改操作并发进行，读操作不加锁;因为HashEntry的value变量是volatilede , 也能保证读取到最新的值；默认将 hash表分为16个桶，get、put、

remove等常用操作只所著对应的桶，所以效率相对于Hashtable高16倍;

Jdk1.8中，ConcurrentHashMap没有用“锁分段”来实现线程安全，而是使用CAS算法和synchronized来确保线程安全，但是底层segment并没有被删除的；底层数据结构是分段数组+链表+红黑树；类似HashMap

参考：<https://www.cnblogs.com/yaoyaoo/p/14381493.html>

<https://baijiahao.baidu.com/s?id=1666189505160307677&wfr=spider&for=pc>

