[**大公司最喜欢问的Java集合类面试题**](http://www.codeceo.com/article/java-collection-interview.html)

2015-03-03    分类：[JAVA开发](http://www.codeceo.com/article/category/develop/java)、[编程开发](http://www.codeceo.com/article/category/develop)、[首页精华](http://www.codeceo.com/article/category/pick)[5人评论](http://www.codeceo.com/article/java-collection-interview.html#comments)     来源：[IT宅男](http://www.jianshu.com/p/4e13fcdf9ab5)

分享到：更多12

看了一些所谓大公司的JAVA面试问题，发现对于JAVA集合类的使用都比较看重似的，而自己在这方面还真的是所真甚少，抽空也学习学习吧。

java.util包中包含了一系列重要的集合类，而对于集合类，主要需要掌握的就是它的内部结构，以及遍历集合的迭代模式。

**接口：Collection**

Collection是最基本的集合接口，一个Collection代表一组Object，即Collection的元素（Elements）。一些Collection允许相同的元素而另一些不行。一些能排序而另一些不行。Java SDK不提供直接继承自Collection的类，Java SDK提供的类都是继承自Collection的“子接口”如List和Set。

所有实现Collection接口的类都必须提供两个标准的构造函数：无参数的构造函数用于创建一个空的Collection，有一个Collection参数的构造函数用于创建一个新的Collection，这个新的Collection与传入的Collection有相同的元素。后一个构造函数允许用户复制一个Collection。

主要的一个接口方法：boolean add(Ojbect c)  
虽然返回的是boolean，但不是表示添加成功与否，这个返回值表示的意义是add()执行后，集合的内容是否改变了（就是元素的数量、位置等有无变化）。类似的addAll，remove，removeAll，remainAll也是一样的。

**用Iterator模式实现遍历集合**

Collection有一个重要的方法：iterator()，返回一个Iterator（迭代器），用于遍历集合的所有元素。Iterator模式可以把访问逻辑从不同的集合类中抽象出来，从而避免向客户端暴露集合的内部结构。典型的用法如下：

Iterator it = collection.iterator(); // 获得一个迭代器

while(it.hasNext()) {

Object obj = it.next(); // 得到下一个元素

}

不需要维护遍历集合的“指针”，所有的内部状态都由Iterator来维护，而这个Iterator由集合类通过工厂方法生成。

每一种集合类返回的Iterator具体类型可能不同，但它们都实现了Iterator接口，因此，我们不需要关心到底是哪种Iterator，它只需要获得这个Iterator接口即可，这就是接口的好处，面向对象的威力。

要确保遍历过程顺利完成，必须保证遍历过程中不更改集合的内容（Iterator的remove()方法除外），所以，确保遍历可靠的原则是：只在一个线程中使用这个集合，或者在多线程中对遍历代码进行同步。

由Collection接口派生的两个接口是List和Set。

**List接口**

List是有序的Collection，使用此接口能够精确的控制每个元素插入的位置。用户能够使用索引（元素在List中的位置，类似于数组下标）来访问List中的元素，这类似于Java的数组。和下面要提到的Set不同，List允许有相同的元素。

除了具有Collection接口必备的iterator()方法外，List还提供一个listIterator()方法，返回一个ListIterator接口，和标准的Iterator接口相比，ListIterator多了一些add()之类的方法，允许添加，删除，设定元素，还能向前或向后遍历。

实现List接口的常用类有LinkedList，ArrayList，Vector和Stack。

**LinkedList类**

LinkedList实现了List接口，允许null元素。此外LinkedList提供额外的get，remove，insert方法在LinkedList的首部或尾部。这些操作使LinkedList可被用作堆栈（stack），队列（queue）或双向队列（deque）。

注意LinkedList没有同步方法。如果多个线程同时访问一个List，则必须自己实现访问同步。一种解决方法是在创建List时构造一个同步的List：  
List list = Collections.synchronizedList(new LinkedList(…));

**ArrayList类**

ArrayList实现了可变大小的数组。它允许所有元素，包括null。ArrayList没有同步。  
size，isEmpty，get，set方法运行时间为常数。但是add方法开销为分摊的常数，添加n个元素需要O(n)的时间。其他的方法运行时间为线性。

每个ArrayList实例都有一个容量（Capacity），即用于存储元素的数组的大小。这个容量可随着不断添加新元素而自动增加，但是增长算法并没有定义。当需要插入大量元素时，在插入前可以调用ensureCapacity方法来增加ArrayList的容量以提高插入效率。

和LinkedList一样，ArrayList也是非同步的（unsynchronized）。

**Vector类**

Vector非常类似ArrayList，但是Vector是同步的。由Vector创建的Iterator，虽然和ArrayList创建的Iterator是同一接口，但是，因为Vector是同步的，当一个Iterator被创建而且正在被使用，另一个线程改变了Vector的状态（例如，添加或删除了一些元素），这时调用Iterator的方法时将抛出ConcurrentModificationException，因此必须捕获该异常。

**Stack 类**

Stack继承自Vector，实现一个后进先出的堆栈。Stack提供5个额外的方法使得Vector得以被当作堆栈使用。基本的push和pop方法，还有peek方法得到栈顶的元素，empty方法测试堆栈是否为空，search方法检测一个元素在堆栈中的位置。Stack刚创建后是空栈。

**Set接口**

Set是一种不包含重复的元素的Collection，即任意的两个元素e1和e2都有e1.equals(e2)=false，Set最多有一个null元素。

很明显，Set的构造函数有一个约束条件，传入的Collection参数不能包含重复的元素。

请注意：必须小心操作可变对象（Mutable Object）。如果一个Set中的可变元素改变了自身状态导致Object.equals(Object)=true将导致一些问题。

**Map接口**

请注意，Map没有继承Collection接口，Map提供key到value的映射。一个Map中不能包含相同的key，每个key只能映射一个value。Map接口提供3种集合的视图，Map的内容可以被当作一组key集合，一组value集合，或者一组key-value映射。

**Hashtable类**

Hashtable继承Map接口，实现一个key-value映射的哈希表。任何非空（non-null）的对象都可作为key或者value。

添加数据使用put(key, value)，取出数据使用get(key)，这两个基本操作的时间开销为常数。

Hashtable通过initial capacity和load factor两个参数调整性能。通常缺省的load factor 0.75较好地实现了时间和空间的均衡。增大load factor可以节省空间但相应的查找时间将增大，这会影响像get和put这样的操作。

使用Hashtable的简单示例如下，将1，2，3放到Hashtable中，他们的key分别是”one”，”two”，”three”：  
Hashtable numbers = new Hashtable();  
numbers.put(“one”, new Integer(1));  
numbers.put(“two”, new Integer(2));  
numbers.put(“three”, new Integer(3));

要取出一个数，比如2，用相应的key：  
Integer n = (Integer)numbers.get(“two”);  
System.out.println(“two = ” + n);

由于作为key的对象将通过计算其散列函数来确定与之对应的value的位置，因此任何作为key的对象都必须实现hashCode和equals方法。hashCode和equals方法继承自根类Object，如果你用自定义的类当作key的话，要相当小心，按照散列函数的定义，如果两个对象相同，即obj1.equals(obj2)=true，则它们的hashCode必须相同，但如果两个对象不同，则它们的hashCode不一定不同，如果两个不同对象的hashCode相同，这种现象称为冲突，冲突会导致操作哈希表的时间开销增大，所以尽量定义好的hashCode()方法，能加快哈希表的操作。

如果相同的对象有不同的hashCode，对哈希表的操作会出现意想不到的结果（期待的get方法返回null），要避免这种问题，只需要牢记一条：要同时复写equals方法和hashCode方法，而不要只写其中一个。

Hashtable是同步的。

**HashMap类**

HashMap和Hashtable类似，不同之处在于HashMap是非同步的，并且允许null，即null value和null key。，但是将HashMap视为Collection时（values()方法可返回Collection），其迭代器操作时间开销和HashMap的容量成比例。因此，如果迭代操作的性能相当重要的话，不要将HashMap的初始化容量设得过高，或者load factor过低。

[**WeakHashMap**](http://www.codeceo.com/article/java-weakhashmap-source.html)**类**

WeakHashMap是一种改进的HashMap，它对key实行“弱引用”，如果一个key不再被外部所引用，那么该key可以被GC回收。

**总结**

* 如果涉及到堆栈，队列等操作，应该考虑用List，对于需要快速插入，删除元素，应该使用LinkedList，如果需要快速随机访问元素，应该使用ArrayList。
* 如果程序在单线程环境中，或者访问仅仅在一个线程中进行，考虑非同步的类，其效率较高，如果多个线程可能同时操作一个类，应该使用同步的类。
* 要特别注意对哈希表的操作，作为key的对象要正确复写equals和hashCode方法。
* 尽量返回接口而非实际的类型，如返回List而非ArrayList，这样如果以后需要将ArrayList换成LinkedList时，客户端代码不用改变。这就是针对抽象编程。