细说 Java 之常用集合类

作者：廖雪峰

线性表，链表，哈希表是常用的数据结构，在进行 Java 开发时，JDK 已经为我们提供了一系列相应的类来实现基本的数据结构。这些类均在 java.util 包中。本文试图通过简单的描述，向读者阐述各个类的作用以及如何正确使用这些类。

```
Collection
├List
│ ├LinkedList
│ ├ArrayList
│ └Vector
│    └Stack
└Set
Map
├Hashtable
└HashMap
```

Collection 接口

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。

如何遍历 Collection 中的每一个元素？不论 Collection 的实际类型如何，它都支持一个 iterator()的方法，该方法返回一个迭代子，使用该迭代子即可逐一访问 Collection 中每一个元素。典型的用法如下：

```
Iterator it = collection.iterator(); // 获得一个迭代子
while(it.hasNext()) {
    Object obj = it.next(); // 得到下一个元素
}
```

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，其迭代子操作时间开销和 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 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。

（参考：Sun JDK1.4.1 API DOC）

作者简介：

廖雪峰，长期从事 J2EE/J2ME 开发，对 Open Source 框架有深入研究。目前廖雪峰创建了 JavaEE 开发网（http://www.javaeedev.com），著有《Spring 2.0 核心技术与最佳实践》一书。