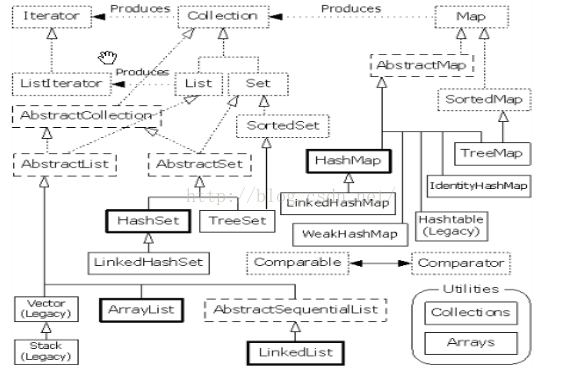
Java集合框架

# 集合框架



说明

点框内的为接口,虚线框内的为抽象类,实线框内的为具体类,粗实线为常用类

虚线空箭头为实现implements

实线空箭头为继承extends(ListIterator和tIterator是继承关系~)

点线实箭头表示可以创建的对象

Queue和List Set同级,规律一致,除了并发应用, Queue的实现有LinkedList和PriorityQueue

规律:

1. Collection,List,Set,Queue,Map都有相应的抽象类AbstractXXX实现接口,作为实现类和接口的中间层, AbstractXXX已经提供相应的基本操作,实现类根据自身小改动即可
2. 对于Set, Map, 都有StoredXXX接口继承,用于Tree的实现

Tree实现使用的算法?

# Map

数组并不保存键本身,而是通过键生成一个散列码作为数组的下标,为散列均匀,数组长度应为质数

|  |  |  |
| --- | --- | --- |
| Type | |  |
| HashMap | 对性能做了优化,最快,可以通过构造器设置容量和负载因子 | |
| LinkedHashMap | 默认以插入顺序访问, 也可以启用按照LRU(Least Recently Use最近最少使用)算法,没有被访问的对象会出现在前面:在访问时,改变结点的引用关系 | |
| TreeMap | 排序,基于红黑树实现,内部对象需要实现Comparable | |
| WeakHashMap | 当key在映射之外没有引用指向key,则可以被垃圾回收器回收 | |
| ConcurrentHashMap | 线程安全 | |
| IdentityHashMap | 比较对象是否相等时,使用==而非equals方法 | |

## HashMap VS HashTable

|  |  |  |
| --- | --- | --- |
|  | HashMap | Hashtable |
| extends | AbstractMap | Dictionary |
| 线程安全/同步 | 否 | 是 |
| Key value 为Null | 允许 | 不允许, NullPointerException |
| fail-fast | Iterator支持 | 实现了Enumeration和Iterator, Iterator方法支持,而Enumeration方法不支持 |

# 底层实现机制

|  |  |  |
| --- | --- | --- |
| Type | 底层实现 | 特点 |
| ArrayList | 数组 |  |
| LinkedList | 双向链表 | LinkedList实现了Queue |
| PriorityQueue | 数组 |  |
| Vector | 数组 | 同步,方法加上了synchronized |
| Stack | 数组 | 同步,方法加上了synchronized  Stack extends Vector(thinking in java 作者吐槽用继承,应该用组合) |
| HashMap | 数组+链表 |  |

# Fail-fast

容器中维护一个modCount的值,用于记录容器修改的次数(remove add rehash),修改一次就++,注意没有—

Java.util包中的所有集合类都被设计为fail-fast的(包括Hashtable,vector,stack等), 而java.util.concurrent中的集合类都为fail-safe的, fail-safe迭代器从不抛出ConcurrentModificationException

当创建迭代器时,记录下当时的modCount

|  |
| --- |
| HashIterator() {  expectedModCount = modCount; // expectedModCount是HashIterator的成员属性  ....  } |

进行迭代操作,如remove时

|  |
| --- |
| public void remove() { //非同步方法  if (modCount != expectedModCount)  throw new ConcurrentModificationException();  ...  HashMap.this.removeEntryForKey(k); //这里更新了modCount  expectedModCount = modCount; //更新记录  } |

通过以上可知,通过迭代器修改时,由于expectedModCount是HashIterator的成员属性,只能记录当前线程下的Iterator实例对象的操作(iterator()方法返回的是新建对象)

而modCount可被多个线程访问,或者被当前线程但不是Iterator实例修改,最终还是会抛出ConcurrentModificationException,所以即使是Collections.synchronizedXXX()包装后的同步类,同样抛异常

正确的做法是自从获得迭代器后,就只能用迭代器去操作,其他操作都会抛ConcurrentModificationException

* **fail-safe**

fail-safe任何对集合结构的修改都会在一个复制的集合上进行修改，因此不会抛出ConcurrentModificationException

The view's iterator is a "weakly consistent" iterator that will never throw ConcurrentModificationException, and guarantees to traverse elements as they existed upon construction of the iterator, and may (but is not guaranteed to) reflect any modifications subsequent to construction.

fail-safe机制有两个问题

（1）需要复制集合，产生大量的无效对象，开销大

（2）无法保证读取的数据是目前原始数据结构中的数据

**Fail Safe Iterator实例**

CopyOnWriteArrayList,ConcurrentHashMap

|  |
| --- |
| /\*\* 储存快照 \*/  private final Object[] snapshot;  private COWIterator(Object[] elements, int initialCursor) {  cursor = initialCursor;  snapshot = elements; //储存快照  } |