**[LinkedHashMap和HashMap的比较使用](http://www.cnblogs.com/hubingxu/archive/2012/02/21/2361281.html)**

由于现在项目中用到了LinkedHashMap，并不是太熟悉就到网上搜了一下。

 import java.util.HashMap;

import java.util.Iterator;

import java.util.LinkedHashMap;

import java.util.Map;

public class TestLinkedHashMap {

  public static void main(String args[])

  {

   System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*LinkedHashMap\*\*\*\*\*\*\*\*\*\*\*\*\*");

   Map<Integer,String> map = new LinkedHashMap<Integer,String>();

   map.put(6, "apple");

   map.put(3, "banana");

   map.put(2,"pear");

   for (Iterator it =  map.keySet().iterator();it.hasNext();)

   {

    Object key = it.next();

    System.out.println( key+"="+ map.get(key));

   }

   System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*HashMap\*\*\*\*\*\*\*\*\*\*\*\*\*");

   Map<Integer,String> map1 = new  HashMap<Integer,String>();

   map1.put(6, "apple");

   map1.put(3, "banana");

   map1.put(2,"pear");

   for (Iterator it =  map1.keySet().iterator();it.hasNext();)

   {

    Object key = it.next();

    System.out.println( key+"="+ map1.get(key));

   }

  }

}

运行结果如下:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*LinkedHashMap\*\*\*\*\*\*\*\*\*\*\*\*\*  
6=apple  
3=banana  
2=pear  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*HashMap\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
2=pear  
6=apple  
3=banana

分析:LinkedHashmap 的特点是put进去的对象位置未发生变化,而HashMap会发生变化.

再普及下：

**java为数据结构中的映射定义了一个接口java.util.Map;它有四个实现类,分别是HashMap Hashtable LinkedHashMap 和TreeMap.**

Map主要用于存储健值对，根据键得到值，因此不允许键重复(重复了覆盖了),但允许值重复。  
Hashmap 是一个最常用的Map,它根据键的HashCode值存储数据（重复存储入相同的hashCode不会导致错误发生，但只会保留一个值，equlas和hashCode的规则，hashCode不同的可能equals。但是不equals一定hashCode不同）,根据键可以直接获取它的值，具有很快的访问速度，遍历时，取得数据的顺序是完全随机的。

HashMap最多只允许一条记录的键为Null;允许多条记录的值为 Null;

HashMap默认按照key的升序排列存储的数据，因此会在一定程度上影响到性能。

HashMap不支持线程的同步，即任一时刻可以有多个线程同时写HashMap;可能会导致数据的不一致。

如果需要同步，可以用 Collections的synchronizedMap方法使HashMap具有同步的能力，或者使用ConcurrentHashMap；但是这样的话性能会有影响。

Hashtable与 HashMap类似,它继承自Dictionary类，不同的是:它不允许记录的键或者值为空;它支持线程的同步，即任一时刻只有一个线程能写Hashtable,因此也导致了 Hashtable在写入时会比较慢。

LinkedHashMap 是HashMap的一个子类，保存了记录的插入顺序，在用Iterator遍历LinkedHashMap时，先得到的记录肯定是先插入的（先进先出，类似栈）.也可以在构造时用带参数，按照应用次数排序。在遍历的时候会比HashMap慢，不过有种情况例外，当HashMap容量很大，实际数据较少时，遍历起来可能会比 LinkedHashMap慢，因为LinkedHashMap的遍历速度只和实际数据有关，和容量无关，而HashMap的遍历速度和他的容量有关。

TreeMap实现SortMap接口，能够把它保存的记录根据键排序,默认是按键值的升序排序，也可以指定排序的比较器，当用Iterator 遍历TreeMap时，得到的记录是排过序的。

一般情况下，我们用的最多的是HashMap,在Map 中插入、删除和定位元素，HashMap 是最好的选择。但如果您要按自然顺序或自定义顺序遍历键，那么TreeMap会更好。如果需要输出的顺序和输入的相同,那么用LinkedHashMap 可以实现,它还可以按读取顺序来排列.

HashMap是一个最常用的Map，它根据键的hashCode值存储数据，根据键可以直接获取它的值，具有很快的访问速度。HashMap最多只允许一条记录的键为NULL，允许多条记录的值为NULL。

HashMap不支持线程同步，即任一时刻可以有多个线程同时写HashMap，可能会导致数据的不一致性。如果需要同步，可以用Collections的synchronizedMap方法使HashMap具有同步的能力。

Hashtable与HashMap类似，不同的是：它不允许记录的键或者值为空；它支持线程的同步，即任一时刻只有一个线程能写Hashtable，因此也导致了Hashtable在写入时会比较慢。

LinkedHashMap保存了记录的插入顺序，在用Iterator遍历LinkedHashMap时，先得到的记录肯定是先插入的。

在遍历的时候会比HashMap慢TreeMap能够把它保存的记录根据键排序，默认是按升序排序，也可以指定排序的比较器。当用Iterator遍历TreeMap时，得到的记录是排过序的。

参考：http://www.cnblogs.com/hubingxu/archive/2012/02/21/2361281.html