# 1.HashMap,HashTable,ConcurrentHashMap底层实现原理和线程安全问题

## HashMap

HashMap是继承了Abstract<K,V>类，实现了Map<K,V>接口,Cloneable接口，Serializable接口的一个类，而Abstract<K,V>类是实现了Map<K,V>接口的抽象类，定义了一个抽象方法entrySet()，返回一个泛型为Entry的Set集合，定义了一个SimpleImmutableEntry<K,V>内部类，这个类实现了Map<K,V>接口中的Entry<K,V>接口，就是一个键值对类，包括get，set等方法。HashMap中定义了一个内部类，Node<K,V>，实现了Map.Entry<K,V>接口，作为键值对类。HashMap是允许key和value为null值。它虽然不是线程安全的，但是它的基础操作get，和put的时间是恒定的，在不需要考虑线程安全的情况下，它的效率会比Hashtable高。

## Hashtable

Hashtable和HashMap是两个很类似得类，除了Hashtable是Synchronized的，以及不允许空值。但是现在已经被淘汰了，因为由于每次操作都要给整个Hashtable加锁的缘故，可能会导致线程阻塞，执行效率较低。

## ConcuurentHashMap

ConcurrentHashMap是线程安全的HashMap的实现，它是利用了分段原理，把map分成了若干个段（ssize，由构造函数中的concurrencyLevel,参数决定），给每个段加锁而不是给整个map加锁，所以每个段就类似与一个Hashtable，然后每个段都有一个table，table的容量由总capacity除以ssize决定，capacity是输入是构造函数的参数，插入值时首先要找出对应的段，然后再找出段中对应的table中的位置，再插入,get时同理。

## HashMap和Hashtable的区别

1. HashMap和Hashtable产生散列冲突时采用的都是链表，即相同hash值的Entry值组成链表。
2. HashMap可以接受值为null的key和value，但是Hashtable不可以。
3. HashMap不是Synchronized,而Hashtable是，它大部分提供给外部的接口方法都是Synchronized.所以Hastable是线程安全的，而HashMap不是。
4. HashMap的迭代器是fail-fast的，就是说如果有一个HashMap对象创建了itrator,那么，如果其他对象试图从结构上更改这个对象，也就是增加或删除一个元素，就会报异常，而Hashtable的迭代器是enumerator迭代器，是可以改变的。

# 2.HashMap的遍历方式

**import** java.util.HashMap;

**import** java.util.Iterator;

**import** java.util.Map;

**import** java.util.Scanner;

**public** **class** Test3 {

//关于HashMap的遍历方式有6种，分别是Map.entrySet(),Map.keySet(),Map.valueSet()的值调用和iterator调用

//下面显示的是Map.entrySet()的iterator调用的泛型参数方法实现

**public** **static** **void** main(String[] args) {

Scanner in=**new** Scanner(System.***in***);

Map<String,String> map=**new** HashMap<>();

String s="";

**for**(**int** i=0;i<5;i++) {

s=in.nextLine();

map.put(s.split(",")[0],s.split(",")[1]);

}

*printHashMap*(map);

}

**public** **static** <T> **void** printHashMap(Map<T,T> map) {

Iterator iter=map.entrySet().iterator();

**while**(iter.hasNext()) {

Map.Entry entry=(Map.Entry)iter.next();

T key=(T) entry.getKey();

T value=(T) entry.getValue();

System.***out***.println("key:"+key+" value:"+value);

}

}

}

# 3.内部比较器和外部比较器的比较

## 使用场景：

TreeSet在添加元素时会根据它的内部比较器来排序元素，所以在使用TreeSet这样的容器时就使用内部比较器。

List不会自动排序，但可以通过Collections类来传入外部比较器进行排序，所以使用外部比较器更好。