# HashTable底层原理

## 6.1 类的层级关系

HashTable<K, V>继承自Dictionary<K, V>实现Map<K, V>, Cloneable, Serializable

## 6.2 类的相关属性

HashTable的容器（桶）



table的total总数



HashTable的扩容因子



HashTable的加载因子，扩容因子=总容量\*加载因子



修改次数



## 6.3 HashTable和HashMap的区别

1. HashMap是线程不安全的。HashTable是线程安全的，使用synchronized修饰。
2. HashTable的key和value均不允许为null，否则抛出空指针。HashMap的key和value都允许为null。
3. HashTable默认初始容量是11，加载因子是0.75f。HashMap的默认初始容量是16，加载因子是0.75f。
4. 如果超过扩容阈值，新的容量大小是【（原数组容量\*2） + 1】。
5. HashTable已经过时，因为有性能问题，如果想线程安全的使用HashMap，可以考虑使用JUC的ConcurrentHashMap。ConcurrentHashMap使用的是分段锁并不会整个访问都锁定，所以会不HashTable性能好很多。
6. Hashtable、HashMap都使用了 Iterator。而由于历史原因，Hashtable还使用了Enumeration的方式 。
7. HashMap的Iterator是fail-fast迭代器。当有其它线程改变了HashMap的结构（增加，删除，修改元素），将会抛出ConcurrentModificationException。不过，通过Iterator的remove()方法移除元素则不会抛出ConcurrentModificationException异常。但这并不是一个一定发生的行为，要看JVM。JDK8之前的版本中，Hashtable是没有fast-fail机制的。在JDK8及以后的版本中 ，HashTable也是使用fast-fail的， modCount的使用类似于并发编程中的CAS（Compare and Swap）技术。我们可以看到这个方法中，每次在发生增删改的时候都会出现modCount++的动作。而modcount可以理解为是当前hashtable的状态。每发生一次操作，状态就向前走一步。设置这个状态，主要是由于hashtable等容器类在迭代时，判断数据是否过时时使用的。尽管hashtable采用了原生的同步锁来保护数据安全。但是在出现迭代数据的时候，则无法保证边迭代，边正确操作。于是使用这个值来标记状态。一旦在迭代的过程中状态发生了改变，则会快速抛出一个异常，终止迭代行为。

## 6.4 HashTable的构造函数

和HashMap的构造函数类似。默认容量为11，加载因子为0.75f

