参考：<http://blog.csdn.net/u012926924/article/details/50452411>

**modCount到底是干什么的呢**

原创 2016年01月03日 20:26:06

* 标签：
* [线程安全](http://so.csdn.net/so/search/s.do?q=%E7%BA%BF%E7%A8%8B%E5%AE%89%E5%85%A8&t=blog) /
* [线程](http://so.csdn.net/so/search/s.do?q=%E7%BA%BF%E7%A8%8B&t=blog) /
* [hashmap](http://so.csdn.net/so/search/s.do?q=hashmap&t=blog)
* 7220

modCount到底是干什么的呢

在ArrayList,LinkedList,HashMap等等的内部实现增，删，改中我们总能看到modCount的身影，modCount字面意思就是修改次数，但为什么要记录modCount的修改次数呢？   
大家发现一个公共特点没有，所有使用modCount属性的全是线程不安全的，这是为什么呢？说明这个玩意肯定和线程安全有关系喽，那有什么关系呢

阅读源码，发现这玩意只有在本数据结构对应迭代器中才使用,以HashMap为例：

private abstract class HashIterator<E> implements Iterator<E> {

Entry<K,V> next; // next entry to return

int expectedModCount; // For fast-fail

int index; // current slot

Entry<K,V> current; // current entry

HashIterator() {

expectedModCount = modCount;

if (size > 0) { // advance to first entry

Entry[] t = table;

while (index < t.length && (next = t[index++]) == null)

;

}

}

public final boolean hasNext() {

return next != null;

}

final Entry<K,V> nextEntry() {

if (modCount != expectedModCount)

throw new ConcurrentModificationException();

Entry<K,V> e = next;

if (e == null)

throw new NoSuchElementException();

if ((next = e.next) == null) {

Entry[] t = table;

while (index < t.length && (next = t[index++]) == null)

;

}

current = e;

return e;

}

public void remove() {

if (current == null)

throw new IllegalStateException();

if (modCount != expectedModCount)

throw new ConcurrentModificationException();

Object k = current.key;

current = null;

HashMap.this.removeEntryForKey(k);

expectedModCount = modCount;

}

}

由以上代码可以看出，在一个迭代器初始的时候会赋予它调用这个迭代器的对象的mCount，如何在迭代器遍历的过程中，一旦发现这个对象的mcount和迭代器中存储的mcount不一样那就抛异常   
好的，下面是这个的完整解释   
**Fail-Fast 机制**   
我们知道 java.util.HashMap 不是线程安全的，因此如果在使用迭代器的过程中有其他线程修改了map，那么将抛出ConcurrentModificationException，这就是所谓fail-fast策略。这一策略在源码中的实现是通过 modCount 域，modCount 顾名思义就是修改次数，对HashMap 内容的修改都将增加这个值，那么在迭代器初始化过程中会将这个值赋给迭代器的 expectedModCount。在迭代过程中，判断 modCount 跟 expectedModCount 是否相等，如果不相等就表示已经有其他线程修改了 Map：注意到 modCount 声明为 volatile，保证线程之间修改的可见性。

所以在这里和大家建议，当大家遍历那些非线程安全的数据结构时，尽量使用迭代器