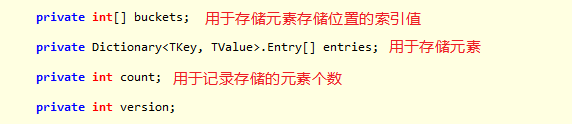
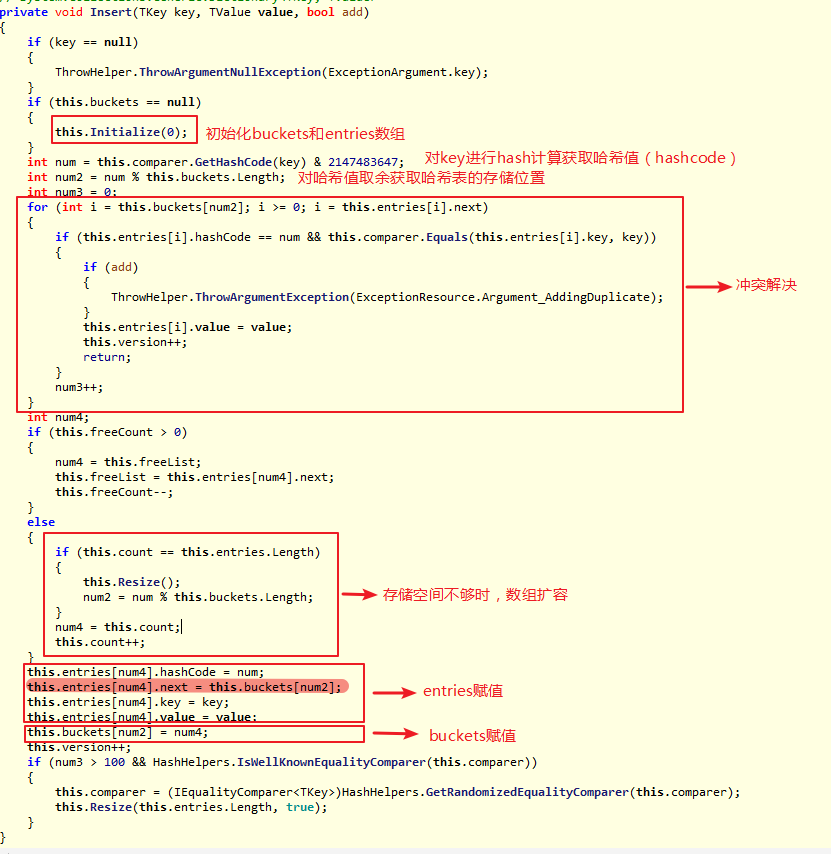
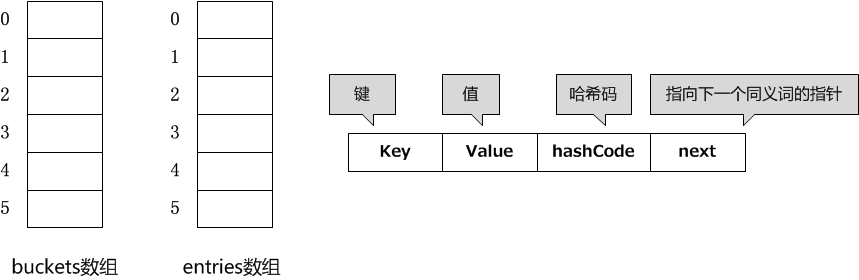
**Dictionary<TKey,TValue>**

**Dictionary<TKey,TValue>**的查询数据所花费的时间是所有集合类里面最快的，因为其内部使用了散列函数加双数组来实现，所以其查询数据操作的时间复杂度可以认为是O（1）。**Dictionary<TKey,TValue>**的实现是一种典型的**牺牲空间换取时间**（双数组）的做法。

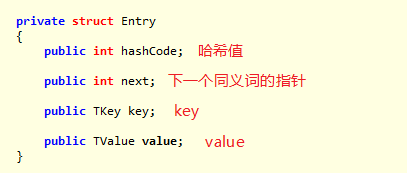


**Dictionary<TKey,TValue>添加新元素的实现：**

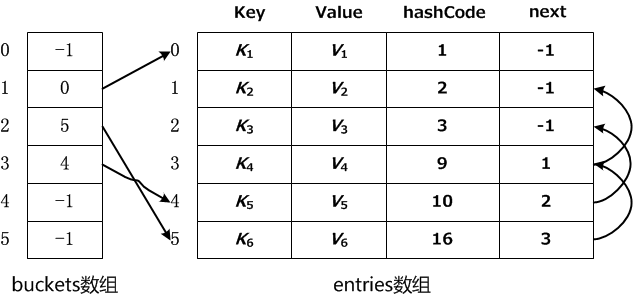




**Dictionary<TKey,TValue>**内部有两个数组，一个数组名为**buckets**，用于存放由多个同义词组成的静态链表头指针（链表的第一个元素在数组中的索引号，当它的值为-1时表示此哈希地址不存在元素）；另一个数组为**entries**，它用于存放哈希表中的实际数据，同时这些数据通过next指针构成多个单链表。**entries**数组中所存放的是Entry结构体，Entry结构体由4个部分组成，如下所示：

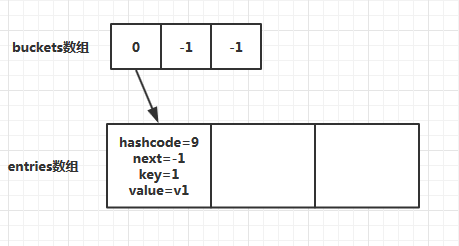


**Dictionary<TKey,TValue>**计算key的哈希值使用的是取余法，这种方式可能会产生冲突，所以需要进行冲突解决。**Dictionary<TKey,TValue>**解决冲突的方式是链接法。

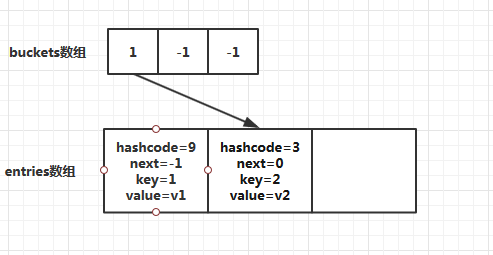


我们可以根据源码来模拟推导一下这个过程：

当添加第一个元素时，此时会分配哈希表**buckets**数组和**entries**数组的空间和初始大小，默认为3。对key=1进行哈希求值，假设第一个元素的哈希值=9，然后targetBucket = 9%buckets.Length(3)的值为0，所以第一个元素应该放在**entries**数组的第一位。最后对哈希表**buckets**数组赋值，数组索引为0，值为0。此时内部结构如图所示：

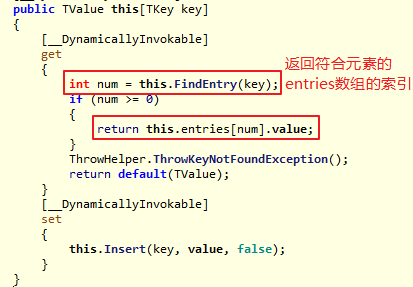


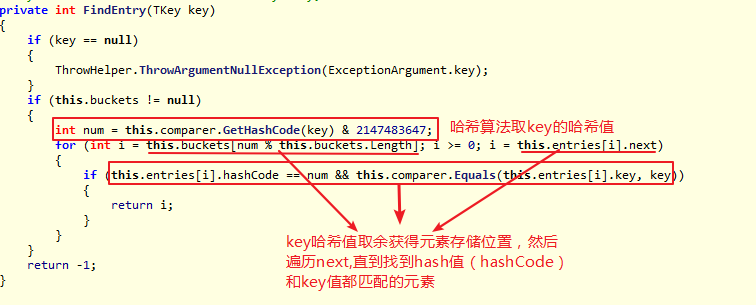
然后插入第二个元素，对key=2进行哈希求值，假设第二个元素的哈希值=3，然后targetBucket = 3%buckets.Length (默认是3)的值为0，所以第二个元素应该放在**entries**数组的第一位。但是**entries**数组的第一位已经存在元素了，这就发生了冲突。**Dictionary<TKey,TValue>**解决冲突的方式是链接法，把发生冲突的元素链接之前元素的后面，通过next属性来指定冲突关系，最后更新哈希表**buckets**数组。此时内部结构如图所示：



我们可以通过**Dictionary<TKey,TValue>**查找元素的实现来证明我们上面的分析是正确的。

**Dictionary<TKey,TValue>查找元素的实现：**





**Dictionary<TKey,TValue>**之所以能实现快速查找元素，其内部使用哈希表来存储元素对应的位置，我们可以通过哈希值快速地从哈希表中定位元素所在的位置索引，从而快速获取到key对应的Value值。物极必反，**Dictionary<TKey,TValue>**的缺点也很明显，就是里面的数据是无序排列的，所以按照一定顺序遍历查找数据效率是非常低的。