**哈希表**

哈希表（Hashtable）又称为“散列”，Hashtable是会根据索引键的哈希程序代码组织成的索引键（Key）和值（Value）配对的集合。Hashtable 对象是由包含集合中元素的哈希桶（Bucket）所组成的。而[Bucket](http://www.nowamagic.net/academy/tag/Bucket)是Hashtable内元素的虚拟子群组，可以让大部分集合中的搜寻和获取工作更容易、更快速。

哈希函数（Hash Function）为根据索引键来返回数值哈希程序代码的算法。索引键（Key）是被存储对象的某些属性值（Value）。当对象加入至 Hashtable时，它存储在与对象哈希程序代码相符的哈希程序代码相关的Bucket中。当在Hashtable内搜寻值时，哈希程序代码会为该值产生，并且会搜寻与该哈希程序代码相关的Bucket。例如，student和teacher会放在不同的Bucket中，而dog和god会放在相同的 Bucket中。所以当索引键是唯一从Hashtable获取元素的性能时表现会较好。Hash的四大优点如下所示。

* 事先不需要排序。
* 搜寻速度与数据多少无关。
* 数字签名的密码技术保密性（Security）高。
* 可做数据压缩（Data Compression），以节省空间。
* Linux内核里的哈希表应用非常广泛，PHP内核里大部分语言特性也是基于哈希表实现的。为什么哈希表能这么神通广大？哈希表能够实现高效的数据存储和查找，而存储和查找是编程中应用最广泛的两个操作。

**Linux内核里的哈希表**

读过Linux内核源码的人可能都会发现，其中并没有太多复杂的数据结构，作为基础数据结构的双向链表（list）和基于list实现的hash表占据了绝大部分数据结构。内核为什么会大量使用这两种数据结构呢？围绕这个问题（主要是hash表），我将以自己的理解揣摩一下其意图。

首先，这两种数据结构都十分简单，简单包括理解起来简单和使用起来简单两方面内容。这也意味着代码的可读性和可维护性都比其他复杂的数据结构要好，出现bug的风险也较低。从哲学上来讲，这也符合K.I.S.S.条款。

其次，内核是一个比较讲究性能的软件，为了程序设计和维护的简单性而失掉性能，这究竟是不是算得不偿失呢？我们是不是应该将天平更加偏向于性能？已经记不起是在哪里听说过，很多商业的路由软件都是基于二叉树的数据结构来存储路由项，以求得其路由查找的时间复杂度为log(n)，并且他批评Linux的路由项组织为hash表，致使性能不佳，不适合商业。确实有一定道理，可仔细分析，hash表的性能真的比二叉树差么？二叉树的插入和删除某一项的时间复杂度都为log(n);hash表插入和删除的时间复杂度最好为O(1)，最差为O(n)，如果选取的表项(m)足够多，且hash函数足够好的话，其时间复杂度为O(n/m)（当m<=n时）。当m > n / log(n)的时候，hash表的平均表现就比二叉树要好;且当m>=n时，其时间复杂度趋近于O(1)。m的值可以做成可调整的，这也正显示了内核的可定制性。不过，不要盲目乐观，这一切都是以一个足够好的hash函数为前期的。

**hash函数的优劣**

如何判定一个hash函数的好坏呢？

hash的中文意思是“散列”，可解释为：分散排列。一个好的hash函数应该做到对所有元素平均分散排列，尽量避免或者降低他们之间的冲突（Collision）。有必要再次提醒大家的是，hash函数的选择必须慎重，如果不幸所有的元素之间都产生了冲突，那么hash表将退化为链表，其性能会大打折扣，时间复杂度迅速降为O(n)，绝对不要存在任何侥幸心理，因为那是相当危险的。历史上就出现过利用Linux内核hash函数的漏洞，成功构造出大量使hash表发生碰撞的元素，导致系统被DoS，所以目前内核的大部分hash函数都有一个随机数作为参数进行掺杂，以使其最后的值不能或者是不易被预测。这又对 hash函数提出了第二点安全方面的要求：hash函数最好是单向的，并且要用随机数进行掺杂。提到单向，你也许会想到单向散列函数md4和md5，很不幸地告诉你，他们是不适合的，因为hash函数需要有相当好的性能。

一筹莫展了吧？谁叫你又想闭门造车了！还是看看前辈们是如何做的，充分发扬拿来主义的精神，我又称这种做法为“不战而屈人之兵”，这难道不是兵家之上上策么？Linux内核里面用的jhash是一个久经考验，并被实践证明经得起考验的hash函数，可以CPMS(Copy Paste Modify Save)之。Jhash的作者Bob Jenkins在其网站上还公布了诸如针对能预知的数据进行hash的hash函数--完美（perfect）hash函数等一系列其他hash函数，看客们可以选择之，如果有兴趣继续钻研，也可以踏在他们的肩膀上。

**什么是bucket**

bucket的英文解释：

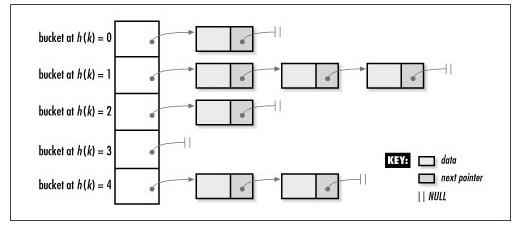
Hash table lookup operations are often O(n/m) (where n is the number of objects in the table and m is the number of buckets), which is close to O(1), especially when the hash function has spread the hashed objects evenly through the hash table, and there are more hash buckets than objects to be stored.

可以这样理解：

一个HASH的结果所对应的地址可存放两个BUCKET。可解决HASH冲突。

* 要存数据时，第一次HASH到这里，在第一个BUCKET存放一个数据。
* 要存数据时，当第二次因某些原因HASH到这里时，在第二个BUCKET存放另一个数据。

一个由5个buckets组成的[哈希表](http://www.nowamagic.net/academy/tag/%E5%93%88%E5%B8%8C%E8%A1%A8)，里面有7个元素：

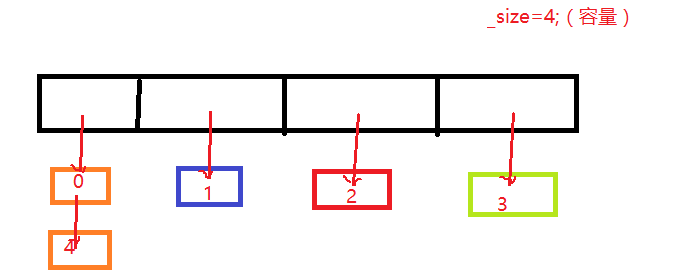


linux的hash函数hash\_long等，用了golden ratio来计算。因为桶(bits)的数量需要由hash函数和对冲突的期望来决定，那么对于hash\_long这样的hash函数，我们怎么确定桶的数量呢？

一般情况下都是自己根据数据特性来考虑使用的 hash 算法，不是千篇一律咬死一个不放。

比如存放 IP 地址的 hash table，用一个 65536 的桶就很好，把 IP 的后 16bit 作为 key。这种方法绝对比 hash\_long、jhash 等函数的碰撞率低。

其实就是这个界和性能的折中。我可以取我问题空间的最大值。这样肯定能保证键值分散。但是这样会浪费很多空间。然而取得太小，又影响查找效率。感觉还是要在试验中进行测试。而且个人觉得，hash比其他搜索的数据结构灵活的地方就是它的可定制性。可以根据具体情况调整，以达到最优的效果。

* **哈希桶**：哈希桶就是盛放不同key链表的容器（即是哈希表），在这里我们可以把每个key的位置看作是一个孔，孔里放了一个链表
* 相信大家可以看出来，使用一个数组来存放记录方法的哈希冲突太多，基于载荷因子的束缚，空间利用率不高，在需要节省空间的情况下，我们可以用哈希桶来处理哈希冲突。
* 哈希桶是使用一个顺序表来存放**具有相同哈希值的key的链表**的头节点，利用这个头节点可以找到其它的key。
* [](http://s1.51cto.com/wyfs02/M01/7F/DD/wKioL1cwU8fCYPvlAAAWi57Xddc860.png)