**[各种字符串Hash函数比较](http://www.byvoid.com/blog/string-hash-compare/" \o "Permanent Link to 各种字符串Hash函数比较)**

[計算機科學](http://www.byvoid.com/blog/cate/computing-science/) [Add comments](http://www.byvoid.com/blog/string-hash-compare/#respond)3,247 views

常用的字符串Hash函数还有ELFHash，APHash等等，都是十分简单有效的方法。这些函数使用位运算使得每一个字符都对最后的函数值产生影响。另外还有以MD5和SHA1为代表的杂凑函数，这些函数几乎不可能找到碰撞。

常用字符串哈希函数有BKDRHash，APHash，DJBHash，JSHash，RSHash，SDBMHash，PJWHash，ELFHash等等。对于以上几种哈希函数，我对其进行了一个小小的评测。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Hash函数 | 数据1 | 数据2 | 数据3 | 数据4 | 数据1得分 | 数据2得分 | 数据3得分 | 数据4得分 | 平均分 |
| BKDRHash | 2 | 0 | 4774 | 481 | 96.55 | 100 | 90.95 | 82.05 | 92.64 |
| APHash | 2 | 3 | 4754 | 493 | 96.55 | 88.46 | 100 | 51.28 | 86.28 |
| DJBHash | 2 | 2 | 4975 | 474 | 96.55 | 92.31 | 0 | 100 | 83.43 |
| JSHash | 1 | 4 | 4761 | 506 | 100 | 84.62 | 96.83 | 17.95 | 81.94 |
| RSHash | 1 | 0 | 4861 | 505 | 100 | 100 | 51.58 | 20.51 | 75.96 |
| SDBMHash | 3 | 2 | 4849 | 504 | 93.1 | 92.31 | 57.01 | 23.08 | 72.41 |
| PJWHash | 30 | 26 | 4878 | 513 | 0 | 0 | 43.89 | 0 | 21.95 |
| ELFHash | 30 | 26 | 4878 | 513 | 0 | 0 | 43.89 | 0 | 21.95 |

其中数据1为100000个字母和数字组成的随机串哈希冲突个数。数据2为100000个有意义的英文句子哈希冲突个数。数据3为数据1的哈希值与1000003(大素数)求模后存储到线性表中冲突的个数。数据4为数据1的哈希值与10000019(更大素数)求模后存储到线性表中冲突的个数。

经过比较，得出以上平均得分。平均数为平方平均数。可以发现，BKDRHash无论是在实际效果还是编码实现中，效果都是最突出的。APHash也是较为优秀的算法。DJBHash,JSHash,RSHash与SDBMHash各有千秋。PJWHash与ELFHash效果最差，但得分相似，其算法本质是相似的。

在信息修竞赛中，要本着易于编码调试的原则，个人认为BKDRHash是最适合记忆和使用的。

BYVoid原创，欢迎建议、交流、批评和指正。

附：各种哈希函数的C语言程序代码

[?](http://www.ericbess.com/ericblog/2008/03/03/wp-codebox/#examples)Download [Hash.c](http://www.byvoid.com/blog/wp-content/plugins/wp-codebox/wp-codebox.php?p=286&download=Hash.c)

|  |
| --- |
| unsigned int SDBMHash(char \*str)  {  unsigned int hash = 0;    while (\*str)  {  // equivalent to: hash = 65599\*hash + (\*str++);  hash = (\*str++) + (hash << 6) + (hash << 16) - hash;  }    return (hash & 0x7FFFFFFF);  }    // RS Hash Function  unsigned int RSHash(char \*str)  {  unsigned int b = 378551;  unsigned int a = 63689;  unsigned int hash = 0;    while (\*str)  {  hash = hash \* a + (\*str++);  a \*= b;  }    return (hash & 0x7FFFFFFF);  }    // JS Hash Function  unsigned int JSHash(char \*str)  {  unsigned int hash = 1315423911;    while (\*str)  {  hash ^= ((hash << 5) + (\*str++) + (hash >> 2));  }    return (hash & 0x7FFFFFFF);  }    // P. J. Weinberger Hash Function  unsigned int PJWHash(char \*str)  {  unsigned int BitsInUnignedInt = (unsigned int)(sizeof(unsigned int) \* 8);  unsigned int ThreeQuarters = (unsigned int)((BitsInUnignedInt \* 3) / 4);  unsigned int OneEighth = (unsigned int)(BitsInUnignedInt / 8);  unsigned int HighBits = (unsigned int)(0xFFFFFFFF) << (BitsInUnignedInt - OneEighth);  unsigned int hash = 0;  unsigned int test = 0;    while (\*str)  {  hash = (hash << OneEighth) + (\*str++);  if ((test = hash & HighBits) != 0)  {  hash = ((hash ^ (test >> ThreeQuarters)) & (~HighBits));  }  }    return (hash & 0x7FFFFFFF);  }    // ELF Hash Function  ELF hash是对字符串进行hash操作时的常用函数 这个算法将一个字符串的数组中的每个元素依次按前四位与上一个元素的低四位相与，组成一个长整形，如果长整的高四位大于零，那么就将它折回再与长整的低四位相异或，要注意最后得到的整型不是唯一的，一开始因为不知道这个吃过亏。  unsigned int ELFHash(char \*str)  {  unsigned int hash = 0;  unsigned int x = 0;    while (\*str)  {  hash = (hash << 4) + (\*str++);  if ((x = hash & 0xF0000000L) != 0)  {  hash ^= (x >> 24);  hash &= ~x;  }  }    return (hash & 0x7FFFFFFF);  }    // BKDR Hash Function  unsigned int BKDRHash(char \*str)  {  unsigned int seed = 131; // 31 131 1313 13131 131313 etc..  unsigned int hash = 0;    while (\*str)  {  hash = hash \* seed + (\*str++);  }    return (hash & 0x7FFFFFFF);  }    // DJB Hash Function  unsigned int DJBHash(char \*str)  {  unsigned int hash = 5381;    while (\*str)  {  hash += (hash << 5) + (\*str++);  }    return (hash & 0x7FFFFFFF);  }    // AP Hash Function  unsigned int APHash(char \*str)  {  unsigned int hash = 0;  int i;    for (i=0; \*str; i++)  {  if ((i & 1) == 0)  {  hash ^= ((hash << 7) ^ (\*str++) ^ (hash >> 3));  }  else  {  hash ^= (~((hash << 11) ^ (\*str++) ^ (hash >> 5)));  }  }  return (hash & 0x7FFFFFFF);  } |