参考：<http://blog.csdn.net/jason5186/article/details/9037623>

**几种经典的hash算法**

计算理论中，没有Hash函数的说法，只有单向函数的说法。所谓的单向函数，是一个复杂的定义，大家可以去看计算理论或者密码学方面的数据。用“人 类”的语言描述单向函数就是：如果某个函数在给定输入的时候，很容易计算出其结果来；而当给定结果的时候，很难计算出输入来，这就是单项函数。各种加密函 数都可以被认为是单向函数的逼近。Hash函数（或者成为散列函数）也可以看成是单向函数的一个逼近。即它接近于满足单向函数的定义。

Hash函数还有另外的含义。实际中的Hash函数是指把一个大范围映射到一个小范围。把大范围映射到一个小范围的目的往往是为了节省空间，使得数据容易保存。除此以外，Hash函数往往应用于查找上。所以，在考虑使用Hash函数之前，需要明白它的几个限制：

1. Hash的主要原理就是把大范围映射到小范围；所以，你输入的实际值的个数必须和小范围相当或者比它更小。不然冲突就会很多。  
2. 由于Hash逼近单向函数；所以，你可以用它来对数据进行加密。  
3. 不同的应用对Hash函数有着不同的要求；比如，用于加密的Hash函数主要考虑它和单项函数的差距，而用于查找的Hash函数主要考虑它映射到小范围的冲突率。

应用于加密的Hash函数已经探讨过太多了，在作者的博客里面有更详细的介绍。所以，本文只探讨用于查找的Hash函数。

Hash函数应用的主要对象是数组（比如，字符串），而其目标一般是一个int类型。以下我们都按照这种方式来说明。

一般的说，Hash函数可以简单的划分为如下几类：  
1. 加法Hash；  
2. 位运算Hash；  
3. 乘法Hash；  
4. 除法Hash；  
5. 查表Hash；  
6. 混合Hash；  
下面详细的介绍以上各种方式在实际中的运用。

一 加法Hash

所谓的加法Hash就是把输入元素一个一个的加起来构成最后的结果。标准的加法Hash的构造如下：

 static int additiveHash(String key, int prime)  
 {  
  int hash, i;  
  for (hash = key.length(), i = 0; i < key.length(); i++)  
   hash += key.charAt(i);  
  return (hash % prime);  
 }  
 这里的prime是任意的质数，看得出，结果的值域为[0,prime-1]。

二 位运算Hash

这类型Hash函数通过利用各种位运算（常见的是移位和异或）来充分的混合输入元素。比如，标准的旋转Hash的构造如下：

 static int rotatingHash(String key, int prime)  
 {  
   int hash, i;  
   for (hash=key.length(), i=0; i<key.length(); ++i)  
     hash = (hash<<4)^(hash>>28)^key.charAt(i);  
   return (hash % prime);  
 }

先移位，然后再进行各种位运算是这种类型Hash函数的主要特点。比如，以上的那段计算hash的代码还可以有如下几种变形：  
1.     hash = (hash<<5)^(hash>>27)^key.charAt(i);  
2.     hash += key.charAt(i);  
        hash += (hash << 10);  
        hash ^= (hash >> 6);  
3.     if((i&1) == 0)  
        {  
         hash ^= (hash<<7) ^ key.charAt(i) ^ (hash>>3);  
        }  
        else  
        {  
         hash ^= ~((hash<<11) ^ key.charAt(i) ^ (hash >>5));  
        }  
4.     hash += (hash<<5) + key.charAt(i);  
5.     hash = key.charAt(i) + (hash<<6) + (hash>>16) – hash;  
6.     hash ^= ((hash<<5) + key.charAt(i) + (hash>>2));

三 乘法Hash

这种类型的Hash函数利用了乘法的不相关性（乘法的这种性质，最有名的莫过于平方取头尾的随机数生成算法，虽然这种算法效果并不好）。比如，

 static int bernstein(String key)  
 {  
   int hash = 0;  
   int i;  
   for (i=0; i<key.length(); ++i) hash = 33\*hash + key.charAt(i);  
   return hash;  
 }

jdk5.0里面的String类的hashCode()方法也使用乘法Hash。不过，它使用的乘数是31。推荐的乘数还有：131, 1313, 13131, 131313等等。

使用这种方式的著名Hash函数还有：  
 //  32位FNV算法  
 int M\_SHIFT = 0;  
    public int FNVHash(byte[] data)  
    {  
        int hash = (int)2166136261L;  
        for(byte b : data)  
            hash = (hash \* 16777619) ^ b;  
        if (M\_SHIFT == 0)  
            return hash;  
        return (hash ^ (hash >> M\_SHIFT)) & M\_MASK;  
}

以及改进的FNV算法：  
    public static int FNVHash1(String data)  
    {  
        final int p = 16777619;  
        int hash = (int)2166136261L;  
        for(int i=0;i<data.length();i++)  
            hash = (hash ^ data.charAt(i)) \* p;  
        hash += hash << 13;  
        hash ^= hash >> 7;  
        hash += hash << 3;  
        hash ^= hash >> 17;  
        hash += hash << 5;  
        return hash;  
}

除了乘以一个固定的数，常见的还有乘以一个不断改变的数，比如：  
    static int RSHash(String str)  
    {  
        int b    = 378551;  
        int a    = 63689;  
        int hash = 0;

       for(int i = 0; i < str.length(); i++)  
       {  
          hash = hash \* a + str.charAt(i);  
          a    = a \* b;  
       }  
       return (hash & 0x7FFFFFFF);  
}

虽然Adler32算法的应用没有CRC32广泛，不过，它可能是乘法Hash里面最有名的一个了。关于它的介绍，大家可以去看RFC 1950规范。

四 除法Hash

除法和乘法一样，同样具有表面上看起来的不相关性。不过，因为除法太慢，这种方式几乎找不到真正的应用。需要注意的是，我们在前面看到的hash的 结果除以一个prime的目的只是为了保证结果的范围。如果你不需要它限制一个范围的话，可以使用如下的代码替代”hash%prime”： hash = hash ^ (hash>>10) ^ (hash>>20)。

五 查表Hash

查表Hash最有名的例子莫过于CRC系列算法。虽然CRC系列算法本身并不是查表，但是，查表是它的一种最快的实现方式。查表Hash中有名的例子有：Universal Hashing和Zobrist Hashing。他们的表格都是随机生成的。

六 混合Hash

混合Hash算法利用了以上各种方式。各种常见的Hash算法，比如MD5、Tiger都属于这个范围。它们一般很少在面向查找的Hash函数里面使用。

七 对Hash算法的评价

<http://www.burtleburtle.net/bob/hash/doobs.html> 这个页面提供了对几种流行Hash算法的评价。我们对Hash函数的建议如下：

1. 字符串的Hash。最简单可以使用基本的乘法Hash，当乘数为33时，对于英文单词有很好的散列效果（小于6个的小写形式可以保证没有冲突）。复杂一点可以使用FNV算法（及其改进形式），它对于比较长的字符串，在速度和效果上都不错。

2. 长数组的Hash。可以使用<http://burtleburtle.net/bob/c/lookup3.c>这种算法，它一次运算多个字节，速度还算不错。

八 后记

本文简略的介绍了一番实际应用中的用于查找的Hash算法。Hash算法除了应用于这个方面以外，另外一个著名的应用是巨型字符串匹配（这时的 Hash算法叫做：rolling hash，因为它必须可以滚动的计算）。设计一个真正好的Hash算法并不是一件容易的事情。做为应用来说，选择一个适合的算法是最重要的。

**九 数组hash**

inline int hashcode(const int \*v)  
{  
int s = 0;  
for(int i=0; i<k; i++)  
s=((s<<2)+(v[i]>>4))^(v[i]<<10);  
s = s % M;  
s = s < 0 ? s + M : s;  
return s;  
}

注：虽说以上的hash能极大程度地避免冲突，但是冲突是在所难免的。所以无论用哪种hash函数，都要加上处理冲突的方法。

原文：http://zhaohaolin.iteye.com/blog/1874420