# 用于工业控制系统的 HASH 算法①

## Hash Arithmetic for Industry Control System

张双伟 (华北计算机系统工程研究所 北京 100083)

**摘 要:** 为满足大容量实时数据库对访问请求的快速响应，需要设计一种高效的内存查询算法。在分析现有 HASH 算法的特点之后，结合实时数据库访问的特点，重新设计了一种 HASH 算法，经过测试，重新设计的 HASH 算法，比现有算法具有更高效率。

**关键词:** HASH 算法 内存查询算法

## 1 引言

工业控制中，实时数据库中的经常要查询的字符串，称为点名，又称位号，一般由大写字符，下划线和数字组成。工业控制中需要控制监视的点越多，字符串就越长。点名字符串由工程人员根据工程需要确定，长度不确定，而且没有具体规律。实时数据库需要对这些字符串进行大量的查询操作。因此查询算法的好坏对实时数据库的性能有重要影响。

## 2 查询算法分析与设计

目前，查询算法有：线性查找算法，时间复杂度为 O(n)；二分法和 B+ Tree 算法，这两种算法时间复杂度为 O(log2N)；HASH 算法，在理想情况下，时间复杂度为 O(1)，但每种 HASH 算法都其使用的不同领域。一种 HASH 算法在不同领域，时间效率差异较大。线性查找算法、二分查找算法和 B+ Tree 算法都随查选数量的增加，时间复杂度成级数增长，因此设计的方向是适应工业控制中使用的字符串的 HASH 算法。

HASH 算法，是根据关键码值直接进行访问的数据结构，也就是说，它通过把关键码值映射到表中一个位置来访问记录，以加快查找的速度。这个映射函数叫做散列函数，存放记录的数组叫做散列表。对不同点名字符串，经 HASH 函数计算 HASH 值时，可能产生相同的 HASH 值，对于 HASH 值相同的点名字符串使用链表解决冲突，HASH 表结构如图 1 所示。

HASH 函数是整个 HASH 算法的关键，HASH 函数应该运算速度快，产生的 HASH 值尽可能均匀分布。

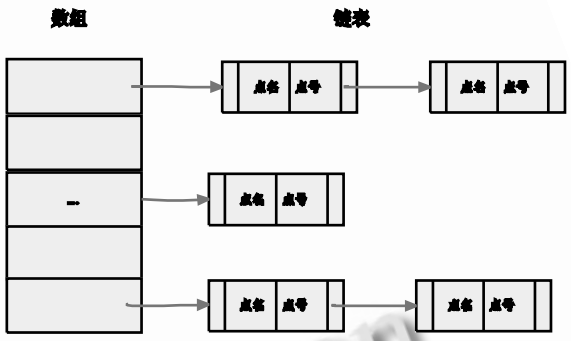


图 1 HASH 结构

### 2.1 HASH 算法工作原理

根据要插入的字符串数，为 HASH 分配数组空间，形成图 1 左侧的数组结构，数组的每个元素为一个指向含有查询关键字(点名)、数据(点号)的节点的指针，并在 HASH 表中记录数组大小。

插入数据：利用 HASH 函数计算关键字的 HASH 值，利用在 HASH 值作为数组下表，在该数组元素指向的链表中，查找指定的关键字是否存在，若不存在申请新节点，插入该链表中；若存在，则返回关键字对应的数据。

查询数据：利用 HASH 函数计算关键字的 HASH 值，利用在 HASH 值作为数组下表，在该数组元素指

① 收稿时间:2009-01-11

向的链表中，查找指定的关键字是否存在，若存在返回数据(点名)，若关键字在 HASH 表中不存在，则返回失败。

释放HASH表:HASH表不再使用后,要释放HASH表占有空间，先释放节点空间，再释放数据空间。

### 2.2 几种经典的 HASH 函数

几款经典软件中使用到的字符串 Hash 函数实现。为使计算的 HASH 值落在一个根据字符串数开辟的 HASH 链表数组的范围内对 HASH 值进行了取余处理。

#### 2.2.1 PHP 中出现的字符串 Hash 函数

PHP 是一种脚本语言，广泛用于网页开发中。该算法每取一个字符访问一次内存，使用移位，异或，位与等运算，每个字符都可以对 HASH 值产生影响。算法如下:

```
int PJWHash( hash_t *tptr,  char* str)
{
    unsigned int hash                = 0;
    unsigned int test                = 0;
   short length=strlen(str);
   short i;
    //对所有字符用左右移位、位与计算 HASH 值
    for( i = 0; i < length; i++)
    {
        hash = (hash << 4) + str[i];
        if((test = hash & 0xF0000000)  != 0)
        {
            hash = (( hash ^ (test >> 24)) & (~0xF0000000));
        }
    }
  //进行取余处理
       return  ((hash  &  0x7FFFFFFF)%tptr->size);
}
```

#### 2.2.2 MySql 中出现的字符串 Hash 函数 1

MYSQL 是一款数据库软件，其中使用的一种 HASH 算法如下，由于计算的 HASH 值存放在 32 位的整数中，算法每次左移 8 位，当前访问的字符计算后放在 HASH 值的低端,每个字符都能影响 HASH 值，但对一个字符要经过位与运算，乘法运算，移位运算等多次运算才能产生影响。

```
int MYSQLHASH1(hash_t *tptr, char*key)
{
  register int nr=1;
  register int nr2=4;
  short length=strlen(key);
 //对所有字符用异或、位与、加法、乘法、左移位计算 HASH 值
  while (length--)
  {    nr^= (((nr & 63)+nr2)*((unsigned int) (char) *key++))+ (nr << 8);
       nr2+=3;
  }
 //进行取余处理
  return ((unsigned int) nr)%tptr->size;
}
```

#### 2.2.3 MySql 中出现的字符串 Hash 函数 2

```
int MYSQLHASH2( hash_t *tptr, char *key)
{
  short len=strlen(key);
 const char *end=key+len;
 unsigned int hash;
 //对所有字符用乘法、异或计算 HASH 值
 for (hash = 0; key < end; key++)
 {
    hash *= 16777619;
    hash ^= (unsigned int) *(char*) key;
 }
 //进行取余处理
 return (hash)%tptr->size;
}
```

MYSQL(数据库软件)中使用的第二种 HASH 算法如上。该算法每个字符都可以对 HASH 值产生影响，但使用乘法运算的次数较多。

#### 2.2.4 一个经典字符串 Hash 函数

```
int EVENThash( hash_t *tptr,  char *key)
{
    char *str=(char*)key;
    register unsigned int h;
    register unsigned char *p;
```

```
        //对所有字符使用乘法计算 HASH 值
        for(h=0, p = (unsigned char *)str; *p ; p++)
            h = 31 * h + *p;
        //进行取余处理
        h=h%tptr->size;
        return h;
    }
```

这种 HASH 算法，每个字符都可以对 HASH 值产生影响，但使用乘法指令的次数较多。

2.2.5 University of Illinois HASH 算法

```
    int RTDBhash(const hash_t *tptr, const char *key) {
        int i=0;
        int hashvalue;
       //对所有字符使用左移位、减法计算 HASH 值
        while (*key != '\0')
          i=(i<<3)+(*key++ - '0');
       //对 HASH 值用乘法、位与运算进行处理，得到适当值
         hashvalue = (((i*1103515249)>>tptr->downshift) & tptr->mask);
        if (hashvalue < 0) {
          hashvalue = 0;
        }
        return hashvalue;
    }
```

这种算法，使用高效的移位指令，每次左移 3 位，但当字符串长度大于 11 时，左边的字符将不影响 HASH 值。

2.2.6 重新设计的 HASH 算法

```
    int ZSWhash(const hash_t *tptr, const char *key,int length)
     {
    int i=0;
    int j=0;
    register int n;
    register int *p=(int*)key;
    //每 4 个字符用乘法、减法计算一次 HASH 值
    for ( n=length;*p&&n>3;n=n-4)
    {
        i=(i*11)+*p++ - 808460784;
    }
    //对剩余字符进行处理
    if (*p!=0)
    {
        switch(n)
        {
        case 1:
            /对剩余的 1 字符进行处理
            i=(i*11)+*(char*)p-48;
            break;
        case 2:
        /对剩余的 2 字符进行处理
            i=(i*11)+*(short*)p-12336;
            break;
        case 3:
            /对剩余的 3 字符进行处理
            memcpy(&j,p,n) ;
            i=(i*11)+j-3158064;
            break;
        default:
            break;
        }
    }
    return (((i*46540617)>>tptr->downshift) & tptr->mask);
    }
```

## 3 测试结果

测试的字符串由大写字母、数字和下划线组成，测试字符串由算法随机生成，生成字符串算法如下：

```
Char
    SYMBOLSET[]={'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z','0','1','2','3','4','5','6','7','8','9','_'};
    #define VARLENGTH 30
    Int symnumber=sizeof(SYMBOLSET);
    for (int i=0;i<要生成的字符串数;i++)
        {
            memset(chline,0,200);
            int j=0;
```

```
            int a=rand()%VARLENGTH;
    //产生一个有大字符 和数字组成的字符串
                for   (;j<(m_PNameLength-2-a);
j++)
            {
                //选取随机字符
                int k = rand()%(symnumber);
                chline[j]=SYMBOLSET[k];
            }
            chline[j]=0;
        …
        …
    }
```

表 1　测试数据

| 测试内容 | 字符长度 30-60，650000 中查找 600000 | | 字符长度 60，650000 中查找 600000，不同字符串末尾 5 个字符相同 | |
|---|---|---|---|---|
| 算法 | 耗时（ms） | 改进 | 耗时(ms) | 改进 |
| PJWhash | 757.613325 | 44.70% | 1058.310414 | 59.30% |
| MYSQLBASE1 | 565.574534 | 25.90% | 610.975382 | 29.40% |
| MYSQLBASE2 | 564.612861 | 25.80% | 626.773068 | 31.30% |
| EVENThash | 496.400233 | 15.60% | 557.431368 | 22.80% |
| RTDBhash | 426.536466 | 1.86% | 1117.277444 | 61.50% |
| ZSWhash | 418.587962 | | 490.102511 | |

在联想开天 M6600，CPU：Intel(R) Pentium (R),D 845@3.00GHZ 双核；内存：512，硬盘：80G 个人电脑上，测试字符长度 30-60，650000 中查找 600000 和字符长度 60，650000 中查找 600000，且不同字符串末尾 5 个字符相同的工程，测试结果见表 1。

## 4　结束语

重新设计的 HASH 算法在字符串长度在 15-30 之间时与 University of Illinois HASH 算法效率几乎相同，而且略高，但当字符串末尾含有 5 个相同字符时，University of Illinois HASH 算法效率明显降低，在以上测试中,重新设计的算法在用于工业控制中的字符串查找时比其他 HASH 算法效率高，达到了重新设计的目的。

## 参考文献

1 严蔚敏.数据结构(C 语言版).北京:清华大学出版社，1997:251－262.

2 Son SH. DRDB: A Distributed Real-Time Database Server for High-Assurance Time-Critical Applications. IEEE, 0730-3157/97,362－367.

3 Pokorny J. Design of V4DB - Experimental Real- Time Database System. IEEE, 1-4244-0136-4/06, 126－131.