<https://blog.csdn.net/yankai0219/article/details/8185816>

转载请注明出处<http://blog.csdn.net/yankai0219/article/details/8185816>

在理论学习篇中，我提到要学会Hash表初始化、插入元素、查找元素三大操作。

在介绍三大操作之前，首先介绍所用到的数据结构。

一、数据结构

     1.hash表的结构

          接下来介绍的都是hash表的拉链法。有两种hash表的结构，推荐使用结构二。详细说明见下面。

          结构一

|  |
| --- |
| //hash table  typedef struct \_HASH\_TABLE{        NODE \* HashTb[HASH\_LEN];  }HASH\_TABLE; |

          结构二

|  |
| --- |
| struct yk\_hash\_counter\_entry {         unsigned int hit\_count;         unsigned int entry\_count;         struct yk\_hash\_key \*keys;  }; |

     2.hash表的桶的结构，即hash表中链表的结点结构

     结构一

|  |
| --- |
| //node  typedef struct \_NODE{        type           data;         struct \_NODE \* next;  }NODE; |

     结构二

|  |
| --- |
| struct yk\_hash\_key {         type key;         struct yk\_hash\_key \*next;  }; |

     说明：

          1）hash表中链表的结点结构是一样的，即包含数据（未hash过的真实数据）及next指针。

          2）hash表的结构推荐结构二，因为在结构二中，包含了除了链表以外的其他信息，更加方便使用。

那么接下来详细论述一下三大操作。

二.Hash表初始化

     1.固定长度的Hash表初始化

               固定长度的Hash表初始化是最简单的也是最常用的一种初始化方式。

                Hash表初始化:就是为hash表数组分配空间，并都赋值为0.

               给出结构二的Hash表初始化

|  |
| --- |
| void hashtest\_init()  {         int i;          hash\_call\_count = 0;        hlist = ( struct yk\_hash\_counter\_entry \*) malloc ( sizeof( structyk\_hash\_counter\_entry) \*  backet\_len);         if (NULL == hlist)        {              perror( "malloc in hashtest\_init" );               return ;        }         for (i = 0; i < backet\_len; i++)        {              hlist[i].hit\_count = 0;              hlist[i].entry\_count = 0;              hlist[i].keys = NULL;        }  } |

     2.长度动态变化的Hash表初始化

          目前我只在Nginx的Hash实现中看到长度动态变化的Hash表的初始化。其原理是根据要插入的元素的数目，动态调整Hash表数组的长度。

          详见文章：

三、插入元素

          插入元素key的原理:

               记hash函数为hash\_func，要插入元素为key。

               1）对于每一个元素key，首先计算其Hash值，hash\_value=hash\_func(key)。通过hash\_value定位到hash表的第hash\_value个元素。

               2）如果hash表的第hash\_value个元素为空，那么直接插入一个结点，结点内容就是该key

               3）如果hash表的第hash\_value个元素不为空，那么检查hash表中是否存在相同元素key，如果存在就退出，不存在的话继续执行。

               4）遍历完整个hash表，依旧不存在相同元素key，那么就新建一个结点，结点内容是该key，然后将该结点插入到链表的末尾。

           函数实现：

|  |
| --- |
| STATUS insert\_data\_into\_hash(HASH\_TABLE \* HashTb,type data)  {         unsigned long key;        NODE \* pNode,\*pNewNode;         if (HashTb == NULL)               return FALSE;        key = HashInt(data);         if ((pNode = HashTb->HashTb[key]) == NULL){                 if ((pNewNode = (NODE\*)malloc(sizeof (NODE))) == NULL)                     return FALSE;              memset(pNewNode,0, sizeof (NODE));              pNewNode->data = data;              pNewNode->next = NULL;                HashTb->HashTb[key] = pNewNode;              printf( "insert succeed \n" );               return TRUE;        } /\*if\*/           if ( (find\_data\_from\_hash(HashTb,data)) ){ /\*if data exits in hash table,it should return FALSE\*/              printf( "data %d exits in hash table\n" ,data);               return FALSE;        }           for (;pNode->next;){              pNode = pNode->next;        } /\*for\*/           if ((pNewNode = (NODE\*)malloc(sizeof (NODE))) == NULL)                           return FALSE;        memset(pNewNode,0, sizeof (NODE));        pNewNode->data = data;        pNewNode->next = NULL;          pNode->next = pNewNode; /\*insert data to end of chain\*/         return TRUE;  } |

四、查找元素

           插入元素key的原理:

               记hash函数为hash\_func，要插入元素为key。

               1）对于每一个元素key，首先计算其Hash值，hash\_value=hash\_func(key)。通过hash\_value定位到hash表的第hash\_value个元素。

               2）如果hash表的第hash\_value个元素为空，返回NULL，不存在该结点

               3）如果hash表的第hash\_value个元素不为空，那么检查hash表中是否存在相同元素key，如果存在就返回该结点的地址，不存在的话返回NULL。

|  |
| --- |
| NODE\* find\_data\_from\_hash(HASH\_TABLE \* HashTb,type data)  {           unsigned long key;        NODE \* pNode;           if (HashTb == NULL)               return NULL;        key = HashInt(data);         if ((pNode = HashTb->HashTb[key]) == NULL){              printf( "NO CHAIN\n" );               return NULL;        }         for (;pNode;pNode = pNode->next){               if (data == pNode->data)                     return pNode;        }        printf( "NO Data\n" );         return NULL;  } |

五、总结

          这是Hash的基础内容，要想进一步学习，就必须先学会这些基本内容。

          这部分程序的完整代码，请到下面网址进行下载<http://download.csdn.net/detail/yankai0219/4768791>

转载请注明出处<http://blog.csdn.net/yankai0219/article/details/8185816>