最近在刷题以及做编程练习的作业时经常会用到哈希表，碰到一些想用的函数时每次都看别人的博客，现结合别人的博客对哈希表做个总结。

本篇博客的主要内容如下

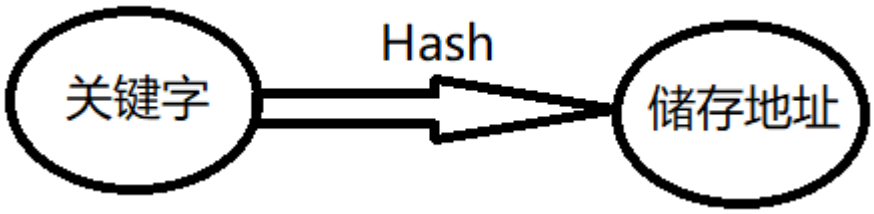
1. 哈希表的定义
2. 如何使用STL库中的哈希表
3. STL中哈希表的常用函数

1. 哈希表 的定义

（1）哈希表的作用

哈希表就是在关键字和存储位置之间建立对应关系，使得元素的查找可以以O(1)的效率进行，其中关键字和存储位置之间是通过散列函数建立关系，记为:

$$Loc(i) = Hash(key_i)。$$



(2) 常见的散列 函数

1）线性定址法：直接取关键字的某个线性函数作为存储地址，散列函数为：

$$Hash(key) = a \times key + b$$

2）除留余数法：将关键字对某一小于散列表长度的数p取余的结果作为存储地址，散列函数为：

$$Hash(key) = key \bmod p$$

3）平方取中法：对关键字取平方，然后将得到结果的中间几位作为存储地址；

4）折叠法：将关键字分割为几部分，然后将这几部分的叠加和作为存储地址。

(3) 地址冲突解决方法

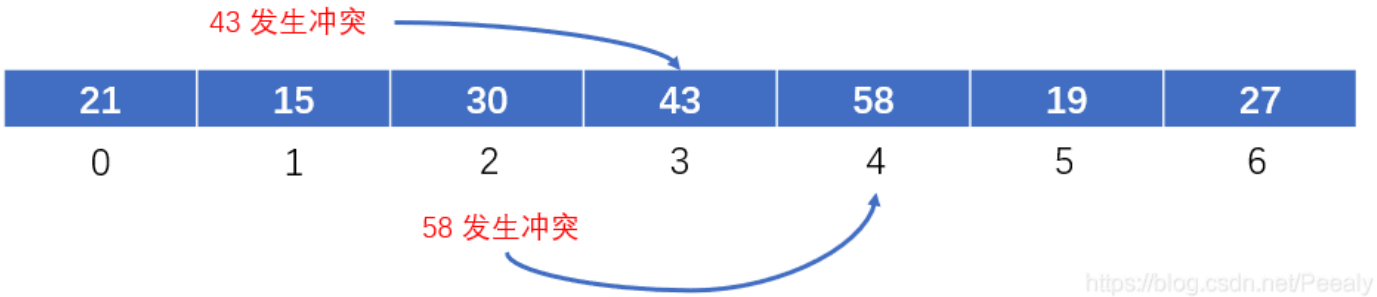
通过以上方法构建的哈希表在理想的情况下能够做到一个关键字对应一个地址，但是实际情况是会有冲突发生，也就是散列函数会将多个关键字映射到同一个地址上。以下是一些解决冲突的方

法：

1）开放地址法：

①线性探测法：当发生冲突时，就顺序查看下一个存储位置，如果位置为空闲状态，就将该关键字存储在该位置上，如果还是发生冲突，就依次往后查看，当查看到存储空间的末尾时还是找不到空位置，就返回从头开始查看；

关键字：15, 21, 30, 43, 58, 19, 27　　散列函数：Hash(key) = key % 7



②平方探测法：不同于前面线性探测法依次顺序查看下一个位置是否能存储元素，平方探测的规则是以$1^2, -1^2, 2^2, -2^2, \ldots$，探测新的存储位置能否存储元素；

③再散列法：利用两个散列函数，当通过第一个散列函数得到关键字的存储地址发生冲突时，再利用第二个散列函数计算出地址增量，地址计算方式如下：

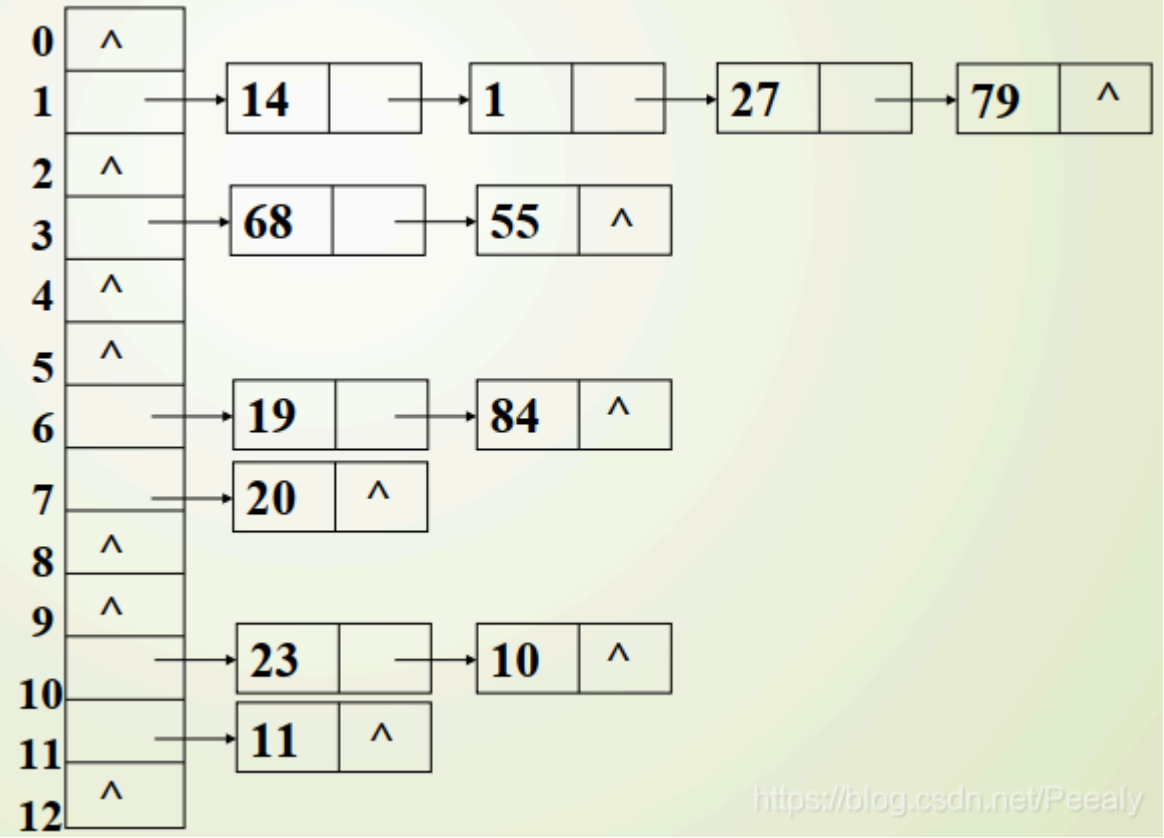
$$H_i = (Hash_1(key) + i * Hash_2(key))\%p$$

④伪随机数法： 当发生地址冲突时，加入一个随机数作为地址增量寻找新的存储地址，地址计算方式如下：

$$H_i = (Hash(key) + d_i)\%p, \text{ 其中} d_i \text{为随机数}$$

2）拉链法

将具有相同存储地址的关键字链成一单链表， m个存储地址就设m个单链表，然后用一个数组将m个单链表的表头指针存储起来，形成一个动态的结构，假设散列函数为 Hash(key) = key %13，其拉链存储结构为：



2. 如何使用STL库中的哈希表

（1）导入头文件

```
#include<unordered_map>
```

（2）哈希表的声明和初始化

1）声明

```
unordered_map<elemType_1, elemType_2> var_name; //声明一个没有任何元素的哈希表，
//其中elemType_1和elemType_2是模板允许定义的类型，如要定义一个键值对都为Int的哈希表：
unordered_map<int, int> map;
```

2）初始化

以上在声明哈希表的时候并没有给unordered_map传递任何参数，因此调用的是unordered_map的默认构造函数，生成一个空容器。初始化主要有一下几种方式：

a）在定义哈希表的时候通过初始化列表中的元素初始化:

```
unordered_map<int, int> hmap{ {1,10},{2,12},{3,13} };
//如果知道要创建的哈希表的元素个数时，也可以在初始化列表中指定元素个数
unordered_map<int, int> hmap{ {{1,10},{2,12},{3,13}},3 };
```

b）通过下标运算来添加元素:

```
//当我们想向哈希表中添加元素时也可以直接通过下标运算符添加元素，格式为：mapName[key]=v
//如：hmap[4] = 14;
//但是这样的添加元素的方式会产生覆盖的问题，也就是当hmap中key为4的存储位置有值时，
//再用hmap[4]=value添加元素，会将原哈希表中key为4存储的元素覆盖
hmap[4] = 14;
hmap[5] = 15;
cout << hmap[4];  //结果为15
```

c）通过insert()函数来添加元素:

```
//通过insert()函数来添加元素的结果和通过下标来添加元素的结果一样，不同的是insert()可以
//insert()函数在同一个key中插入两次，第二次插入会失败
hmap.insert({ 5,15 });
hmap.insert({ 5,16 });
cout << hmap[5];  //结果为15
```

d）复制构造，通过其他已初始化的哈希表来初始新的表:

```
unordered_map<int, int> hmap{ {1,10},{2,12},{3,13} };
unordered_map<int, int> hmap1(hmap);
```

3. STL中哈希表的常用函数

(1) begin( )函数：该函数返回一个指向哈希表开始位置的迭代器

```
unordered_map<int, int>::iterator iter = hmap.begin(); //申请迭代器，并初始化
cout << iter->first << ":" << iter->second;
```

(2) end( )函数：作用于begin函数相同，返回一个指向哈希表结尾位置的下一个元素的迭代器

```
unordered_map<int, int>::iterator iter = hmap.end();
```

(3) cbegin() 和 cend()：这两个函数的功能和begin()与end()的功能相同，唯一的区别是cbegin()和cend()是面向不可变的哈希表

```
const unordered_map<int, int> hmap{ {1,10},{2,12},{3,13} };
unordered_map<int, int>::const_iterator iter_b = hmap.cbegin(); //注意这里的
```

```
unordered_map<int, int>::const_iterator iter_e = hmap.cend();
```

(4) empty()函数：判断哈希表是否为空，空则返回true，非空返回false

```
bool isEmpty = hmap.empty();
```

(5) size()函数：返回哈希表的大小

```
int size = hmap.size();
```

(6) erase()函数： 删除某个位置的元素，或者删除某个位置开始到某个位置结束这一范围内的元素， 或者传入key值删除键值对

```
unordered_map<int, int> hmap{ {1,10},{2,12},{3,13} };
unordered_map<int, int>::iterator iter_begin = hmap.begin();
unordered_map<int, int>::iterator iter_end = hmap.end();
hmap.erase(iter_begin);  //删除开始位置的元素
hmap.erase(iter_begin, iter_end); //删除开始位置和结束位置之间的元素
hmap.erase(3); //删除key==3的键值对
```

(7) at()函数：根据key查找哈希表中的元素

```
unordered_map<int, int> hmap{ {1,10},{2,12},{3,13} };
int elem = hmap.at(3);
```

(8) clear()函数：清空哈希表中的元素

```
hmap.clear()
```

(9) find()函数：以key作为参数寻找哈希表中的元素，如果哈希表中存在该key值则返回该位置上的迭代器，否则返回哈希表最后一个元素下一位置上的迭代器

```
unordered_map<int, int> hmap{ {1,10},{2,12},{3,13} };
unordered_map<int, int>::iterator iter;
iter = hmap.find(2); //返回key==2的迭代器，可以通过iter->second访问该key对应的元
if(iter != hmap.end())  cout << iter->second;
```

(10) bucket()函数：以key寻找哈希表中该元素的储存的bucket编号（unordered_map的源码是基于拉链式的哈希表，所以是通过一个个bucket存储元素）

```
int pos = hmap.bucket(key);
```

(11) bucket_count()函数：该函数返回哈希表中存在的存储桶总数（一个存储桶可以用来存放多个元素，也可以不存放元素，并且bucket的个数大于等于元素个数）

```
int count = hmap.bucket_count();
```

(12) count()函数： 统计某个key值对应的元素个数， 因为unordered_map不允许重复元素，所以返回值为0或1

```
int count = hmap.count(key);
```

(13) 哈希表的遍历: 通过迭代器遍历

```
unordered_map<int, int> hmap{ {1,10},{2,12},{3,13} };
unordered_map<int, int>::iterator iter = hmap.begin();
for( ; iter != hmap.end(); iter++){
 cout << "key: " <<  iter->first  << "value: " <<  iter->second <<endl;
}
```