**0 为什么需要hash\_map**

用过map吧？map提供一个很常用的功能，那就是提供key-value的存储和查找功能。例如，我要记录一个人名和相应的存储，而且随时增加，要快速查找和修改：

岳不群－华山派掌门人，人称君子剑

张三丰－武当掌门人，太极拳创始人

东方不败－第一高手，葵花宝典

...

这些信息如果保存下来并不复杂，但是找起来比较麻烦。例如我要找"张三丰"的信息，最傻的方法就是取得所有的记录，然后按照名字一个一个比较。如果要速度快，就需要把这些记录按照字母顺序排列，然后按照二分法查找。但是增加记录的时候同时需要保持记录有序，因此需要插入排序。考虑到效率，这就需要用到二叉树。讲下去会没完没了，如果你使用STL 的map容器，你可以非常方便的实现这个功能，而不用关心其细节。关于map的数据结构细节，感兴趣的朋友可以参看[学习STL map, STL set之数据结构基础](http://stl.winterxy.com/html/000039.html" \t "_top)。看看map的实现:

#include <map>

#include <string>

using namespace std;

...

map<string, string> namemap;

//增加。。。

namemap["岳不群"]="华山派掌门人，人称君子剑";

namemap["张三丰"]="武当掌门人，太极拳创始人";

namemap["东方不败"]="第一高手，葵花宝典";

...

//查找。。

if(namemap.find("岳不群") != namemap.end()){

...

}

不觉得用起来很easy吗？而且效率很高，100万条记录，最多也只要20次的string.compare的比较，就能找到你要找的记录;200万条记录事，也只要用21次的比较。

速度永远都满足不了现实的需求。如果有100万条记录，我需要频繁进行搜索时，20次比较也会成为瓶颈，要是能降到一次或者两次比较是否有可能？而且当记录数到200万的时候也是一次或者两次的比较，是否有可能？而且还需要和map一样的方便使用。

答案是肯定的。这时你需要has\_map. 虽然hash\_map目前并没有纳入C++ 标准模板库中，但几乎每个版本的STL都提供了相应的实现。而且应用十分广泛。在正式使用hash\_map之前，先看看hash\_map的原理。

**1 数据结构：hash\_map原理**

这是一节让你深入理解hash\_map的介绍，如果你只是想囫囵吞枣，不想理解其原理，你倒是可以略过这一节，但我还是建议你看看，多了解一些没有坏处。

hash\_map基于hash table（哈希表）。 哈希表最大的优点，就是把数据的存储和查找消耗的时间大大降低，几乎可以看成是常数时间；而代价仅仅是消耗比较多的内存。然而在当前可利用内存越来越多的情况下，用空间换时间的做法是值得的。另外，编码比较容易也是它的特点之一。

其基本原理是：使用一个下标范围比较大的数组来存储元素。可以设计一个函数（哈希函数，也叫做散列函数），使得每个元素的关键字都与一个函数值（即数组下标，hash值）相对应，于是用这个数组单元来存储这个元素；也可以简单的理解为，按照关键字为每一个元素“分类”，然后将这个元素存储在相应“类”所对应的地方，称为桶。

但是，不能够保证每个元素的关键字与函数值是一一对应的，因此极有可能出现对于不同的元素，却计算出了相同的函数值，这样就产生了“冲突”，换句话说，就是把不同的元素分在了相同的“类”之中。 总的来说，“直接定址”与“解决冲突”是哈希表的两大特点。

hash\_map，首先分配一大片内存，形成许多桶。是利用hash函数，对key进行映射到不同区域（桶）进行保存。其插入过程是：

1. 得到key
2. 通过hash函数得到hash值
3. 得到桶号(一般都为hash值对桶数求模)
4. 存放key和value在桶内。

其取值过程是:

1. 得到key
2. 通过hash函数得到hash值
3. 得到桶号(一般都为hash值对桶数求模)
4. 比较桶的内部元素是否与key相等，若都不相等，则没有找到。
5. 取出相等的记录的value。

hash\_map中直接地址用hash函数生成，解决冲突，用比较函数解决。这里可以看出，如果每个桶内部只有一个元素，那么查找的时候只有一次比较。当许多桶内没有值时，许多查询就会更快了(指查不到的时候).

由此可见，要实现哈希表, 和用户相关的是：hash函数和比较函数。这两个参数刚好是我们在使用hash\_map时需要指定的参数。

**2 hash\_map 使用**

**2.1 一个简单实例**

不要着急如何把"岳不群"用hash\_map表示，我们先看一个简单的例子：随机给你一个ID号和ID号相应的信息，ID号的范围是1～2的31次方。如何快速保存查找。

#include <hash\_map>

#include <string>

using namespace std;

int main(){

hash\_map<int, string> mymap;

mymap[9527]="唐伯虎点秋香";

mymap[1000000]="百万富翁的生活";

mymap[10000]="白领的工资底线";

...

if(mymap.find(10000) != mymap.end()){

...

}

够简单，和map使用方法一样。这时你或许会问？hash函数和比较函数呢？不是要指定么？你说对了，但是在你没有指定hash函数和比较函数的时候，你会有一个缺省的函数，看看hash\_map的声明，你会更加明白。下面是SGI STL的声明：

template <class \_Key, class \_Tp, class \_HashFcn = hash<\_Key>,

class \_EqualKey = equal\_to<\_Key>,

class \_Alloc = \_\_STL\_DEFAULT\_ALLOCATOR(\_Tp) >

class hash\_map

{

...

}

也就是说，在上例中，有以下等同关系：

...

hash\_map<int, string> mymap;

//等同于:

hash\_map<int, string, hash<int>, equal\_to<int> > mymap;

Alloc我们就不要取关注太多了(希望深入了解Allocator的朋友可以参看[标准库 STL ：Allocator能做什么](http://www.stlchina.org/twiki/bin/view.pl/Main/STLWhatAllocator))

**2.2 hash\_map 的hash函数**

hash< int>到底是什么样子？看看源码:

struct hash<int> {

size\_t operator()(int \_\_x) const { return \_\_x; }

};

原来是个函数对象。在SGI STL中，提供了以下hash函数：

struct hash<char\*>

struct hash<const char\*>

struct hash<char>

struct hash<unsigned char>

struct hash<signed char>

struct hash<short>

struct hash<unsigned short>

struct hash<int>

struct hash<unsigned int>

struct hash<long>

struct hash<unsigned long>

也就是说，如果你的key使用的是以上类型中的一种，你都可以使用缺省的hash函数。当然你自己也可以定义自己的hash函数。对于自定义变量，你只能如此，例如对于string，就必须自定义hash函数。例如：

struct str\_hash{

size\_t operator()(const string& str) const

{

unsigned long \_\_h = 0;

for (size\_t i = 0 ; i < str.size() ; i ++)

\_\_h = 5\*\_\_h + str[i];

return size\_t(\_\_h);

}

};

//如果你希望利用系统定义的字符串hash函数，你可以这样写：

struct str\_hash{

size\_t operator()(const string& str) const

{

return \_\_stl\_hash\_string(str.c\_str());

}

};

在声明自己的哈希函数时要注意以下几点：

1. 使用struct，然后重载operator().
2. 返回是size\_t
3. 参数是你要hash的key的类型。
4. 函数是const类型的。

如果这些比较难记，最简单的方法就是照猫画虎，找一个函数改改就是了。

现在可以对开头的"岳不群"进行哈希化了  . 直接替换成下面的声明即可：

map<string, string> namemap;

//改为：

hash\_map<string, string, str\_hash> namemap;

其他用法都不用边。当然不要忘了吧str\_hash的声明以及头文件改为hash\_map。

你或许会问：比较函数呢？别着急，这里就开始介绍hash\_map中的比较函数。

**2.3 hash\_map 的比较函数**

在map中的比较函数，需要提供less函数。如果没有提供，缺省的也是less< Key> 。在hash\_map中，要比较桶内的数据和key是否相等，因此需要的是是否等于的函数:equal\_to< Key> 。先看看equal\_to的源码：

//本代码可以从SGI STL

//先看看binary\_function 函数声明，其实只是定义一些类型而已。

template <class \_Arg1, class \_Arg2, class \_Result>

struct binary\_function {

typedef \_Arg1 first\_argument\_type;

typedef \_Arg2 second\_argument\_type;

typedef \_Result result\_type;

};

//看看equal\_to的定义：

template <class \_Tp>

struct equal\_to : public binary\_function<\_Tp,\_Tp,bool>

{

bool operator()(const \_Tp& \_\_x, const \_Tp& \_\_y) const { return \_\_x == \_\_y; }

};

如果你使用一个自定义的数据类型，如struct mystruct, 或者const char\* 的字符串，如何使用比较函数？使用比较函数，有两种方法. 第一种是：重载==操作符，利用equal\_to;看看下面的例子：

struct mystruct{

int iID;

int len;

bool operator==(const mystruct & my) const{

return (iID==my.iID) && (len==my.len) ;

}

};

这样，就可以使用equal\_to< mystruct>作为比较函数了。另一种方法就是使用函数对象。自定义一个比较函数体：

struct compare\_str{

bool operator()(const char\* p1, const char\*p2) const{

return strcmp(p1,p2)==0;

}

};

有了compare\_str，就可以使用hash\_map了。

typedef hash\_map<const char\*, string, hash<const char\*>, compare\_str> StrIntMap;

StrIntMap namemap;

namemap["岳不群"]="华山派掌门人，人称君子剑";

namemap["张三丰"]="武当掌门人，太极拳创始人";

namemap["东方不败"]="第一高手，葵花宝典";

**2.4 hash\_map 函数**

hash\_map的函数和map的函数差不多。具体函数的参数和解释，请参看：[STL 编程手册：Hash\_map](http://www.stlchina.org/twiki/bin/view.pl/Main/STLHashMap" \t "_top)，这里主要介绍几个常用函数。

1. hash\_map(size\_type n) 如果讲究效率，这个参数是必须要设置的。n 主要用来设置hash\_map 容器中hash桶的个数。桶个数越多，hash函数发生冲突的概率就越小，重新申请内存的概率就越小。n越大，效率越高，但是内存消耗也越大。
2. const\_iterator find(const key\_type& k) const. 用查找，输入为键值，返回为迭代器。
3. data\_type& operator[](const key\_type& k) . 这是我最常用的一个函数。因为其特别方便，可像使用数组一样使用。不过需要注意的是，当你使用[key ]操作符时，如果容器中没有key元素，这就相当于自动增加了一个key元素。因此当你只是想知道容器中是否有key元素时，你可以使用find。如果你希望插入该元素时，你可以直接使用[]操作符。
4. insert 函数。在容器中不包含key值时，insert函数和[]操作符的功能差不多。但是当容器中元素越来越多，每个桶中的元素会增加，为了保证效率，hash\_map会自动申请更大的内存，以生成更多的桶。因此在insert以后，以前的iterator有可能是不可用的。
5. erase 函数。在insert的过程中，当每个桶的元素太多时，hash\_map可能会自动扩充容器的内存。但在sgi stl中是erase并不自动回收内存。因此你调用erase后，其他元素的iterator还是可用的。

**3 相关hash容器**

hash 容器除了hash\_map之外，还有hash\_set, hash\_multimap, has\_multiset, 这些容器使用起来和set, multimap, multiset的区别与hash\_map和map的区别一样，我想不需要我一一细说了吧。

**4 其他**

这里列几个常见问题，应该对你理解和使用hash\_map比较有帮助。

**4.1 hash\_map和map的区别在哪里？**

* 构造函数。hash\_map需要hash函数，等于函数；map只需要比较函数(小于函数).
* 存储结构。hash\_map采用hash表存储，map一般采用[红黑树(RB Tree)](http://stl.winterxy.com/html/000039.html" \t "_top)实现。因此其memory数据结构是不一样的。

**4.2 什么时候需要用hash\_map，什么时候需要用map?**

总体来说，hash\_map 查找速度会比map快，而且查找速度基本和数据数据量大小，属于常数级别;而map的查找速度是log(n)级别。并不一定常数就比log(n)小，hash还有hash函数的耗时，明白了吧，如果你考虑效率，特别是在元素达到一定数量级时，考虑考虑hash\_map。但若你对内存使用特别严格，希望程序尽可能少消耗内存，那么一定要小心，hash\_map可能会让你陷入尴尬，特别是当你的hash\_map对象特别多时，你就更无法控制了，而且hash\_map的构造速度较慢。

现在知道如何选择了吗？权衡三个因素: 查找速度, 数据量, 内存使用。

这里还有个关于hash\_map和map的小故事，看看:<http://dev.csdn.net/Develop/article/14/14019.shtm>

**4.3 如何在hash\_map中加入自己定义的类型?**

你只要做两件事, 定义hash函数，定义等于比较函数。下面的代码是一个例子：

-bash-2.05b$ cat my.cpp

#include <hash\_map>

#include <string>

#include <iostream>

using namespace std;

//define the class

class ClassA{

public:

ClassA(int a):c\_a(a){}

int getvalue()const { return c\_a;}

void setvalue(int a){c\_a;}

private:

int c\_a;

};

//1 define the hash function

struct hash\_A{

size\_t operator()(const class ClassA & A)const{

// return hash<int>(classA.getvalue());

return A.getvalue();

}

};

//2 define the equal function

struct equal\_A{

bool operator()(const class ClassA & a1, const class ClassA & a2)const{

return a1.getvalue() == a2.getvalue();

}

};

int main()

{

hash\_map<ClassA, string, hash\_A, equal\_A> hmap;

ClassA a1(12);

hmap[a1]="I am 12";

ClassA a2(198877);

hmap[a2]="I am 198877";

cout<<hmap[a1]<<endl;

cout<<hmap[a2]<<endl;

return 0;

}

-bash-2.05b$ make my

c++ -O -pipe -march=pentiumpro my.cpp -o my

-bash-2.05b$ ./my

I am 12

I am 198877

**4.4如何用hash\_map替换程序中已有的map容器？**

这个很容易，但需要你有良好的编程风格。建议你尽量使用typedef来定义你的类型：

typedef map<Key, Value> KeyMap;

当你希望使用hash\_map来替换的时候，只需要修改:

typedef hash\_map<Key, Value> KeyMap;

其他的基本不变。当然，你需要注意是否有Key类型的hash函数和比较函数。

**4.5为什么hash\_map不是标准的？**

具体为什么不是标准的，我也不清楚，有个解释说在STL加入标准C++之时，hash\_map系列当时还没有完全实现，以后应该会成为标准。如果谁知道更合理的解释，也希望告诉我。但我想表达的是，正是因为hash\_map不是标准的，所以许多平台上安装了g++编译器，不一定有hash\_map的实现。我就遇到了这样的例子。因此在使用这些非标准库的时候，一定要事先测试。另外，如果考虑到平台移植，还是少用为佳。