**MYSTL开发文档**

# 1 空间配置器(allocator)

此处的空间配置器采取SGI STL的实现方式，完成一个符合STL规范的版本allocator和一个效率更高的版本alloc。

## allocator版本

allocator版本位于文件allocator.h中。

根据STL的规范，以下是标准allocator的必要接口(来自STL源码解析):

1. 各种数据类型

allocator::value\_type

allocator::pointer

allocator::const\_pointer

allocator::reference

allocator::const\_reference

allocator::size\_type(size\_t)

allocator::difference\_type(ptrdiff\_t)

(注：上述可以用using表达式实现)

1. allocator::rebind

(rebind 用于为与正在实现的容器的元素类型不同的类型分配内存。因为如果有两个相关的类型，例如T和U，此时希望让这两个类型使用同一种分配器，就需要使用rebind；此时的allocator<U>就相当于allocator<T>::rebind<U>::other；所以rebind应该是一个模板结构，只有一个typedef allocator<U> other)

1. 默认构造函数(allocator::allocator()) noexcept ：什么都不用做
2. copy构造函数(allocator::allocator(const allocator& rhs)) noexcept

直接用默认的就可以 = default

1. 泛化的copy构造函数

(是一个模板函数 template<class U>allocator::allocator(const allocator<U>& rhs，可以根据对象U生成对象T) ：什么都不用做

1. 析构函数(allocator::~allocator())

直接使用 = default就可以

1. 返回某个对象的地址

pointer allocator::address(reference x) const

(注: 可以调用const\_pointer来实现非const)

1. 返回某个const对象的地址

const\_pointer allocator::address(const\_reference x) const

1. 配置空间

Pointer allocator::allocate(size\_type n, const void\* = 0)

n代表n个对象，不是n个字节空间；第二个参数是个提示，用来增进区域性(不是很懂???)

1. 归还先前配置的空间

void allocator::deallocate(pointer p, size\_type n)

实现上只需要delete p就行了

1. 返回可成功配置的最大量

size\_type allocator::max\_size() const

这里返回的对象个数，不是字节数；实现是用unsigned int的最大值(-1)除以对象的大小再转换为size\_type类型就可以

1. construct函数

template<class U, class… \_types>

void construct(U\* p, \_types&&… args){  
 ::new (static\_cast<void\*> p) U(::std::forward<\_types>(args)…);

}

用了一个模板类来实现，其中用到了可变参数模板和完美转发技术(可以上网找具体的相关资料)

1. destroy函数

template<class U>

void destroy(U\* p){  
 p->~U();

}

用了一个模板类来实现

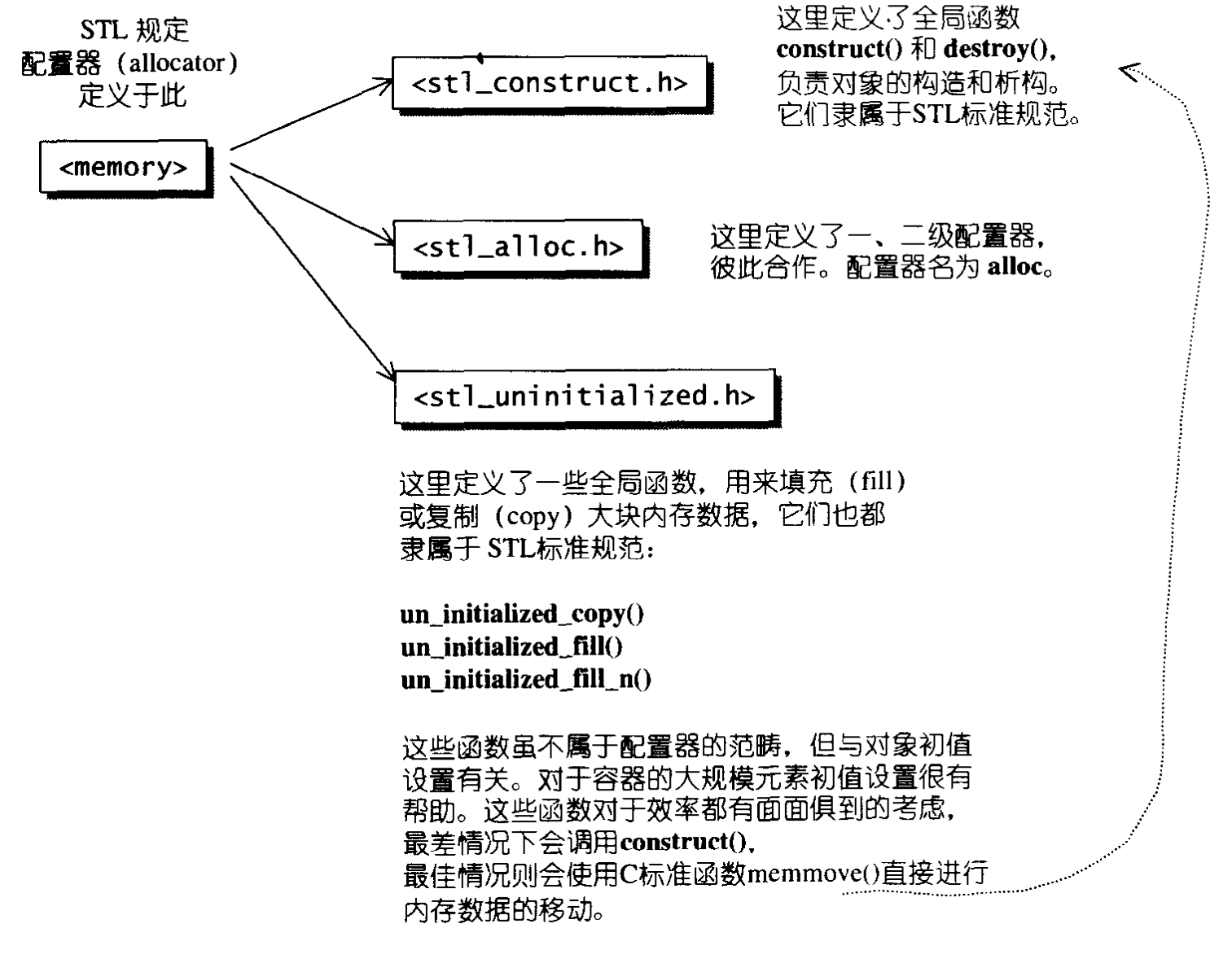
## SGI STL版本

<memory>: 规定SGI STL版本的配置器定于在此文件

<mystl\_construct.h>: 定义了全局函数construct和destory，负责对象的构造和析构

<mystl\_alloc.h>: 定义了一、二级配置器

<stl\_uninitialized.h>: 定义了一些全局函数，用来填充或复制大块内存数据，包括(un\_initialized\_copy、un\_initialized\_fill、un\_initialized\_fill\_n)。这些函数对于效率有面面俱到的考虑，最差情况下会调用construct，最佳情况会使用C标准函数memmove直接进行内存数据的移动



memory文件中也会包含auto\_ptr的实现。

### <mystl\_construct.h>

因为要使用placement new操作，所有要包含new.h头文件

1. construct函数

有两个版本，均为template模板形式，并且因为实现简单，使用inline方式；第一个版本有两个参数，第一个参数为所要构造对象的指针，第二个参数为要构造对象的实值(const + &)；第二个版本的参数只有所要构造对象的指针，采用默认构造的方式进行构造

注意：使用placement new之后记得把指针转换为void\*

1. destory函数

均以template模板形式，并且因为实现简单，使用inline方式；type\_traits在代码中为std::is\_trivially\_destructible<>；\_\_true\_type和\_\_false\_type分别为std::true\_type和std::false\_type；针对char\*，int\*，long\*，float\*，double\*的特化版本什么都不用实现

### <mystl\_alloc.h>

当配置区块大于128bytes时，调用第一级配置器，否则使用内存池来实现内存分配。配置器会主动将任何小型区块的大小调整至8的倍数，并用16个指针来维护，各自管理的大小分别为{8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96, 104, 112, 120, 128}bytes。

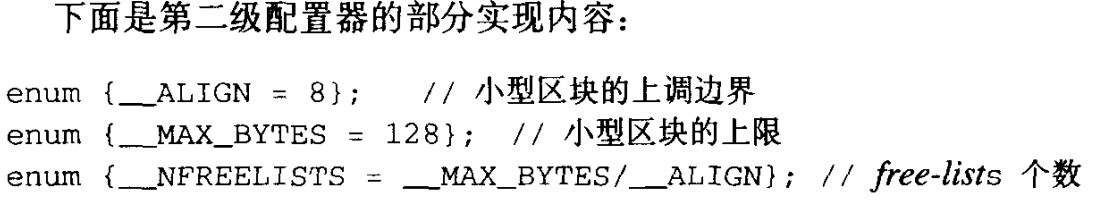
free\_lists的结构如下：

union obj{

union obj\* free\_list\_link;

char user\_data[1];

}



以下是alloc的具体实现：

class{

private:

union obj{

union obj\* free\_list\_link;

char user\_data[1];

}

static char\* start\_free; // 内存池起始位置

static char\* end\_free; // 内存池结束位置

static size\_t heap\_size; // 申请堆空间时附加值的大小

static obj\* free\_list[NFreeList]; // 16个freelist

static size\_t ROUND\_UP(size\_t bytes); // 将大小调整至8的倍数

static size\_t FREELIST\_INDEX(size\_t bytes); // 根据区块大小决定使用第几个freelist

static void\* refill(size\_t n);

static char\* chunk\_alloc(size\_t size, int& nobjs);

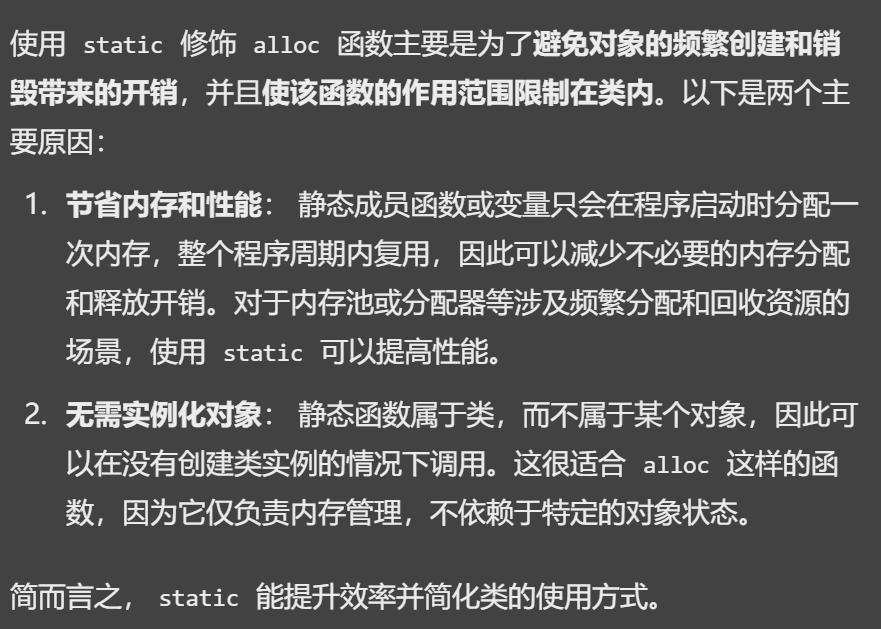
public:

static void\* allocate(size\_t n);

static void deallocate(void\* p, size\_t n);

static void\* reallocate(void\*p, size\_t old\_size, size\_t new\_size);

}



1. ROUND\_UP函数

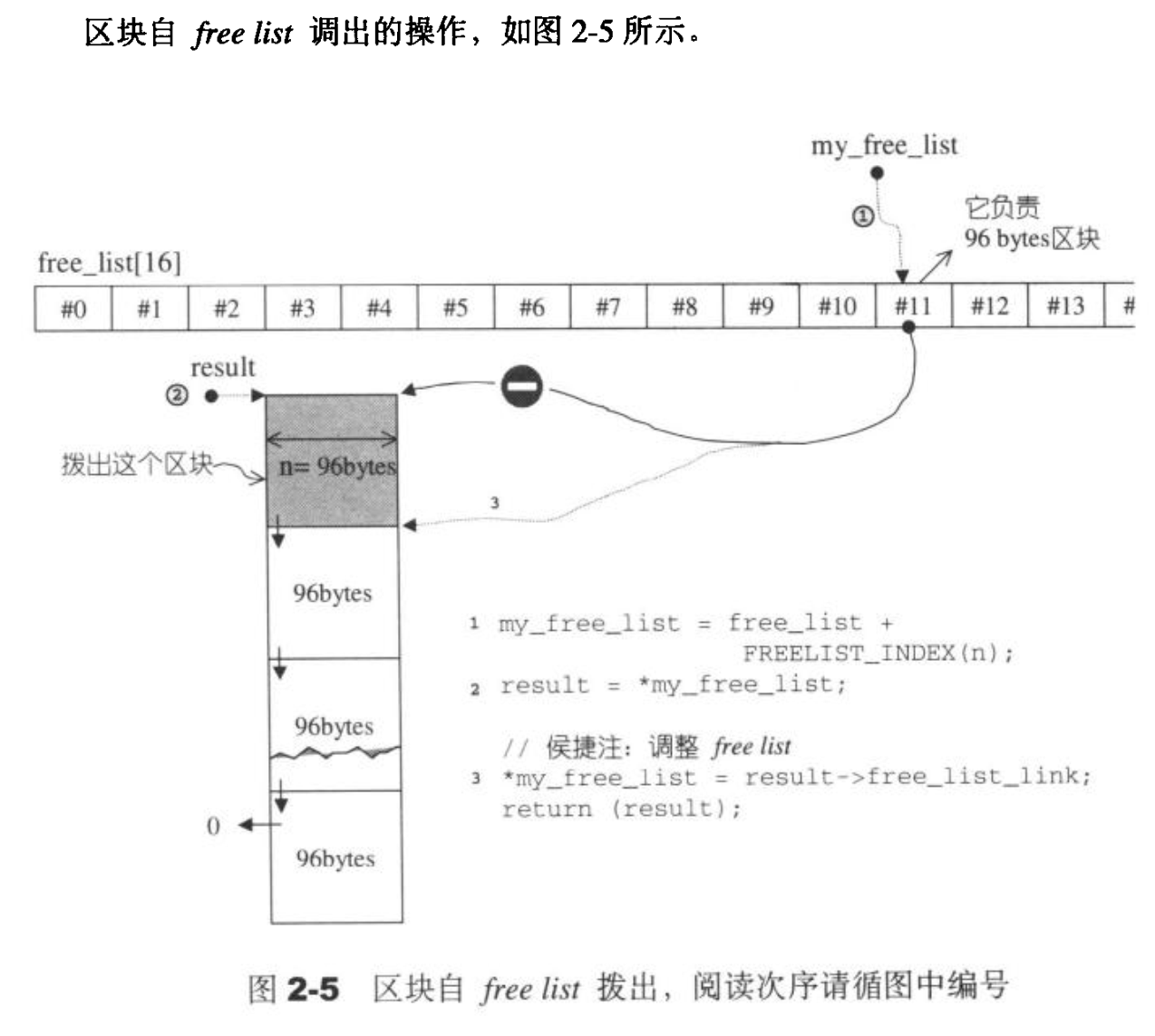
函数的思想就是将传入的参数bytes + 7，然后将最后三个bit清零；具体实现用位操作运算实现。

1. FREELIST\_INDEX函数

根据传入的大小判断属于哪一个指针(8->0，9->1)注意判断参数要大于0以及边界的判断。

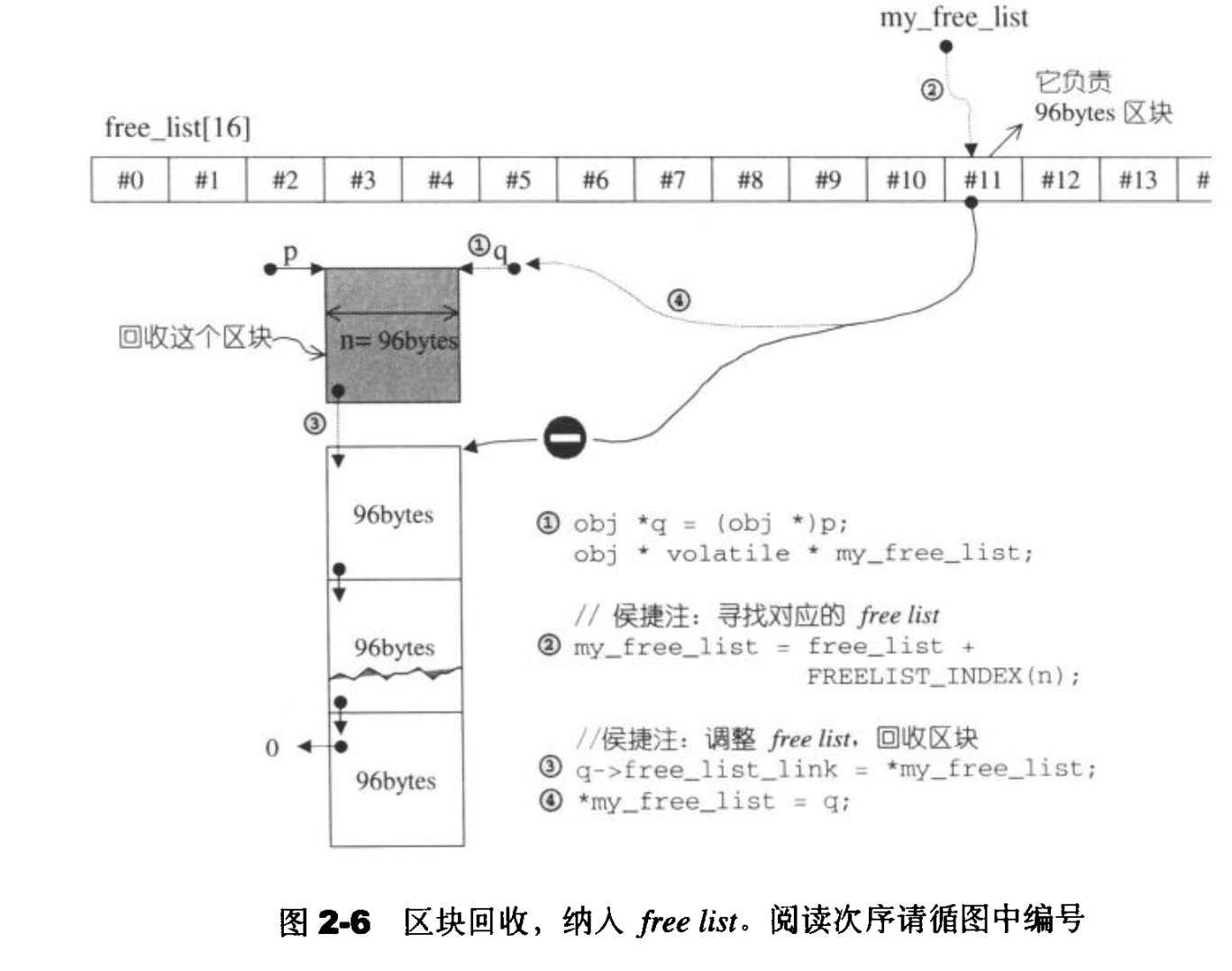
1. 空间配置函数allocate

当申请内存大小大于128bytes时直接调用malloc函数；否则检查参数对应的free\_list中是否有可用的区块，如果有则直接拿过来用，否则就将区块大小上调至8的倍数，再用refill函数填充该区块。



1. 空间释放函数deallocate

如果要回收的空间大小大于128bytes，则直接调用free函数进行回收；否则根据参数大小判断出要回收的大小应该在哪个free\_list上，然后利用头插法进行回收。



1. reallocate函数

直接调用deallocate函数和allocate函数就可以，将原来的空间释放再重新分配新的空间。

1. refill函数

该函数为没有可用空间的free\_list填充空间，新的空间取自内存池中，默认取得20个区块大小，如果内存池的空间不足则可能小于20；如果取得的空间只有一个区块，则直接将该区块返回给用户，否则将剩下的区块连接起来。

注：这里传入的区块个数为引用传参

1. chunk\_alloc函数
2. 内存池的大小完全满足需求量

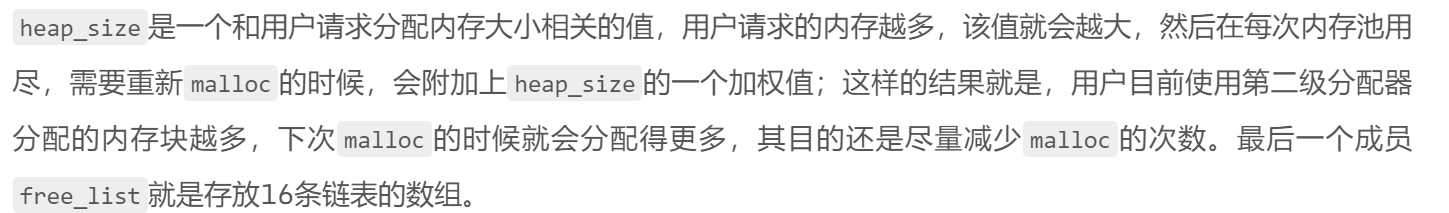
修改内存池的指针并将结果返回。

1. 内存池的大小满足一个及以上的区块大小

将能够分配的空间全部分配并调整内存池的指针。

1. 内存池的大小一个区块大小都不能满足

如果内存池中还剩下有空间，将剩下的空间加入到合适的free\_list中；然后计算出此次需要向heap空间中申请的空间大小(20个区块用于返回，20个区块用于当做内存池，还有额外的空间ROUND\_UP(heap\_size>>4))；如果申请堆空间失败，则搜索在free\_list中有足够大的区块能够分配；如果没有则提示申请空间失败，将end\_free设为null并抛出bad\_alloc异常；如果有则调整start\_free和end\_free并递归调用chunk\_alloc函数分配空间；如果申请堆空间成功则累加heap\_size，调整start\_free和end\_free并递归调用chunk\_alloc分配空间。



### simple\_alloc

由于alloc的配置空间的单位都是bytes，而stl标准的规定是元素的个数，因此需要实现一个simple\_alloc类符合stl标准。simple\_alloc是一个模板，其中有四个成员函数，分别是两个allocate和两个deallocate，分别接收一个或多个元素的空间申请或释放，其实现也是直接调用参数中分配器的allocate和deallocate函数实现。

### <mystl\_uninitialized.h>

这个文件中包含三个函数，作用于未初始化空间上，分别是uninitialized\_copy，uninitialized\_fill，uninitialized\_fill\_n。

1. uninitialized\_copy

template<class InputIterator, class ForwardIterator>

ForwordIterator uninitialized\_copy(InputIterator first, InputIterator last, ForwardIterator result);

实现上根据ForwardIterator是否有trivial copy 赋值操作符来判断使用什么样的动作(std::is\_trivially\_copy\_assignable)；如果是true\_type则直接调用copy函数，如果是false\_type则用for循环调用construct函数。由于此操作必须是原子操作，所以要用try和catch语句实现，如果发生异常则依次调用destroy函数。针对char\*可以直接调用memset函数，因此可以实现出一个针对char\*的特化版本。最后返回的是result的最后一个位置。

1. uninitialized\_fill

template<class ForwordIterator, class T>

ForwardIterator uninitialized\_fill(ForwardIterator first, ForwardIterator last, const T& value)

实现上和uninitialized\_copy一样，只不过true\_type调用的fill函数。返回值为void。

1. uninitialized\_fill\_n

template<class ForwordIterator, class Size, class T>

ForwardIterator uninitialized\_fill\_n(ForwardIterator first, Size n, const T& value)

实现上和uninitialized\_copy一样，只不过true\_type调用的fill\_n函数。返回值为ForwardIterator。

## temporary\_buffer

1. temporary\_buffer是一个模板类，用于临时缓冲区的申请和释放；有两个模板参数ForwardIterator和T；private属性下有三个变量original\_len(ptrdiff\_t)，len(ptrdiff\_t)和buffer(T\*)。
2. 构造和析构函数
3. 构造函数：接受两个参数first和last；调用distance函数算出first和last之间的距离len，调用allocate\_buffer函数，如果len大于0，调用initialize\_buffer函数，如果空间分配失败，释放buffer的空间，将buffer设为空指针，len设为0(这里用try和catch实现)。
4. 复制构造函数：声明为private属性。
5. operator=：声明为private属性。
6. 析构函数：调用destroy函数，释放掉buffer的空间。
7. 迭代器相关函数
8. begin函数：返回buffer；返回值类型为T\*。
9. end函数：返回buffer+len；返回值类型为T\*。
10. 空间大小相关函数
11. requested\_size函数：const成员函数，返回original\_len；返回值类型为ptrdiff\_t。
12. size函数：const成员函数，返回len；返回值类型为ptrdiff\_t。
13. 辅助函数(private属性)
14. allocate\_buffer函数：将original\_len设为len，如果len大于ptrdiff\_t下的最大个数，将len设为最大个数，利用while循环，当len大于0时，调用malloc函数申请空间，如果申请成功退出循环，否则将len减少一半。
15. initialize\_buffer函数：接受两个参数T&类型和std::true\_type；什么都不用做。
16. initialize\_buffer函数：接受两个参数value和false\_type；调用uninitialized\_fill\_n函数对buffer进行填充。

# 2 迭代器

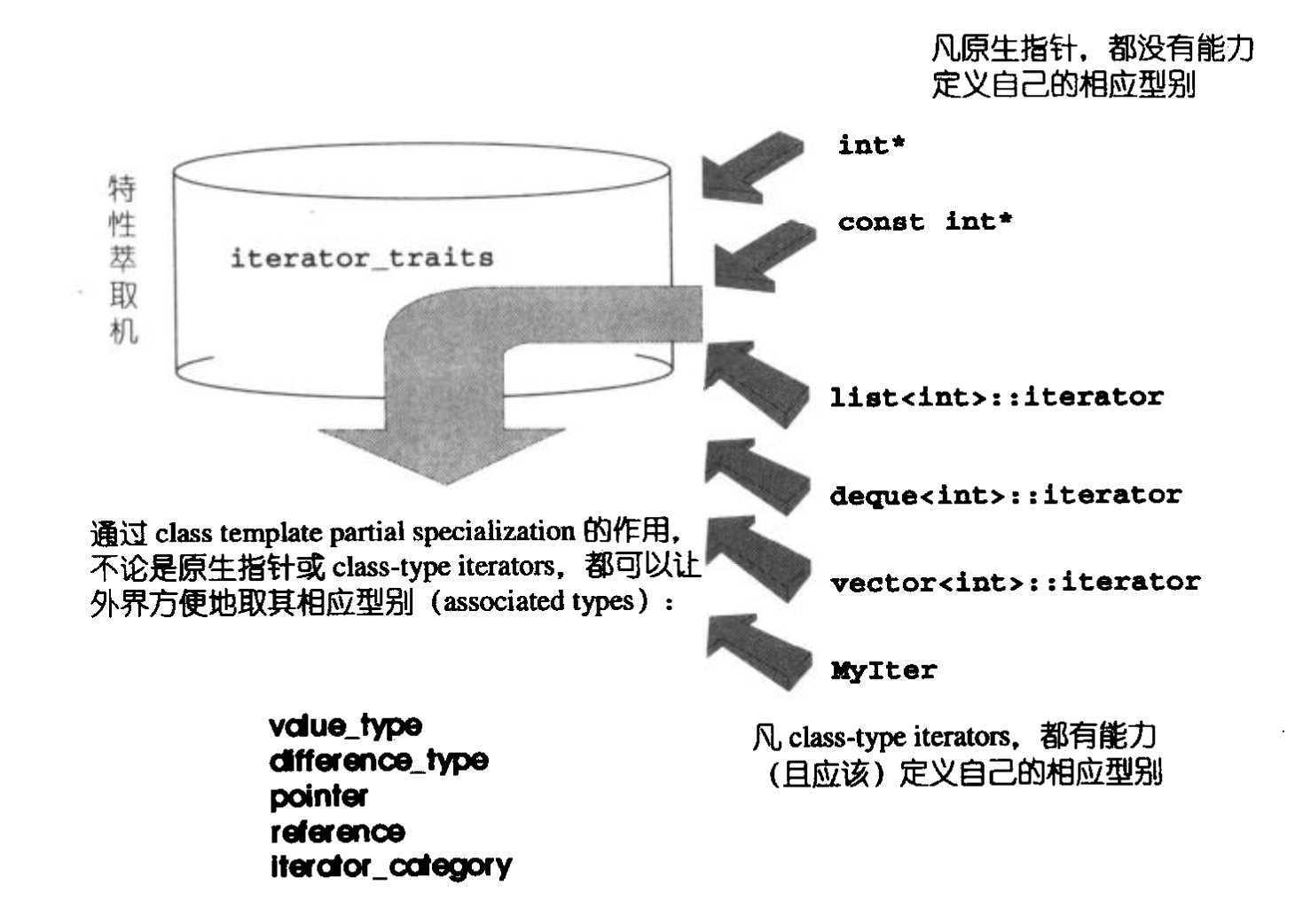
迭代器作为连接容器和算法的桥梁，其最重要的功能就是对operator\*和operator->进行重载。因为auto\_ptr这个模板类实现对上述两个操作符的重载，因此可以先实现出auto\_ptr进而作为参照实现出迭代器。

## auto\_ptr

auto\_ptr作为一个模板类，里面包含了一个实际的指针；构造函数必须显式声明explicit避免隐式转换；它的拷贝构造函数和赋值函数都是以模板的形式实现(如果两个类型不一致的对象进行赋值操作可能会导致编译器报错，在早期这样实现可能会通过编译，但仍然是一种不安全的行为；这应该也是auto\_ptr被抛弃的原因之一吧)。析构函数只需将内部的指针delete；operator\*对内部指针进行解引用并返回引用值；operator->返回当前内部指针；get函数返回内部的指针；release函数通过将内部指针设为空并返回内部指针的副本来达到转移控制权的效果；reset函数先判断参数指针是否和内部指针相等，如果不等则delete内部指针再将内部指针设为参数指针。

## iterator\_traits

通过traits机制可以很好的解决迭代器所指对象类型的问题，并且通过偏特化可以解决原生类型和const类型的问题。



最常用的迭代器相应型别有五种，分别是value type，difference type, reference, pointer, iterator category。

1. value type

迭代器所指对象的类型。

1. difference type

表示两个迭代器之间的距离，也可以用来表示一个迭代器的最大容量。针对原生指针使用c++内建的ptrdiff\_t表示difference type。

1. reference

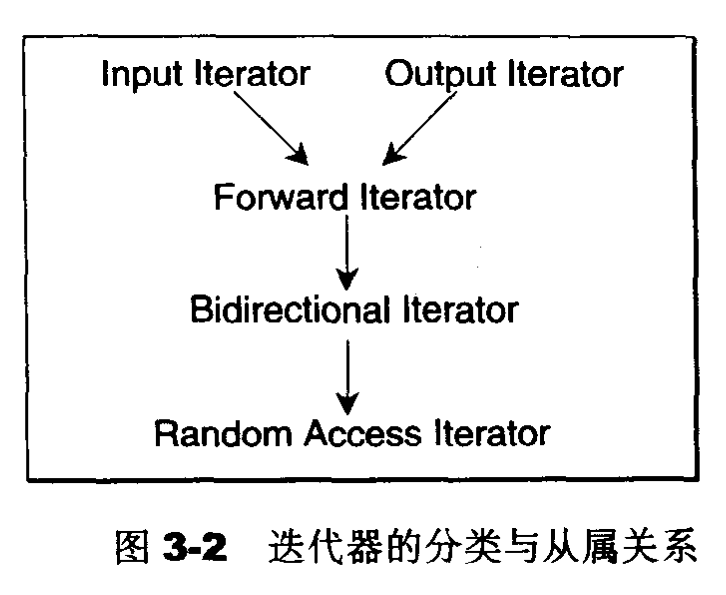
对容器中元素的引用，而不是一个临时对象。

1. pointer

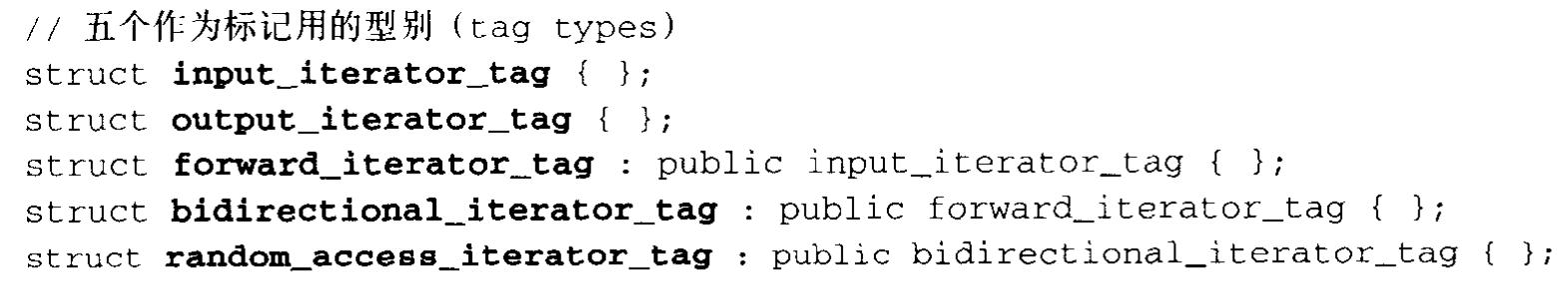
对容器中元素地址的返回。

1. iterator category

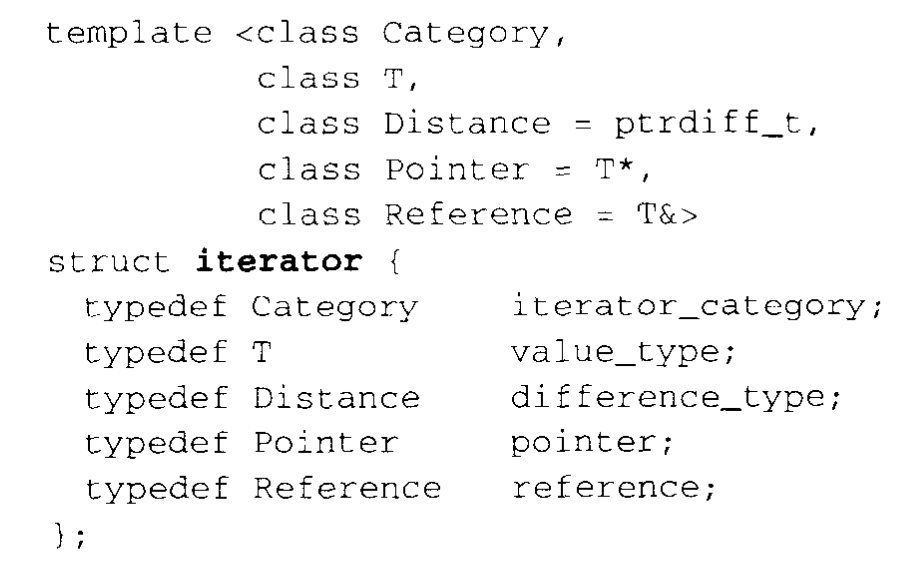
迭代器一共有五种类别，分别是input iterator，output iterator，forward iterator，bidirectional iterator，random iterator。



不同的算法应该选择最合适的迭代器来实现，因此为了重载算法，需要定义五个型别代表五种迭代器类型。



原生指针的型别为random iterator。STL提供了一个iterator类，在设计新的迭代器时只需要继承这个类并提供相应的参数即可。



STL还提供了iterator\_category函数，difference\_type函数，value\_type函数，distance函数和advance函数。

注意：distance\_type和value\_type函数返回的是一个指针，这是因为返回指针可以节省更多的空间(如果类型很大)，并且不需要使用构造函数，节省时间。

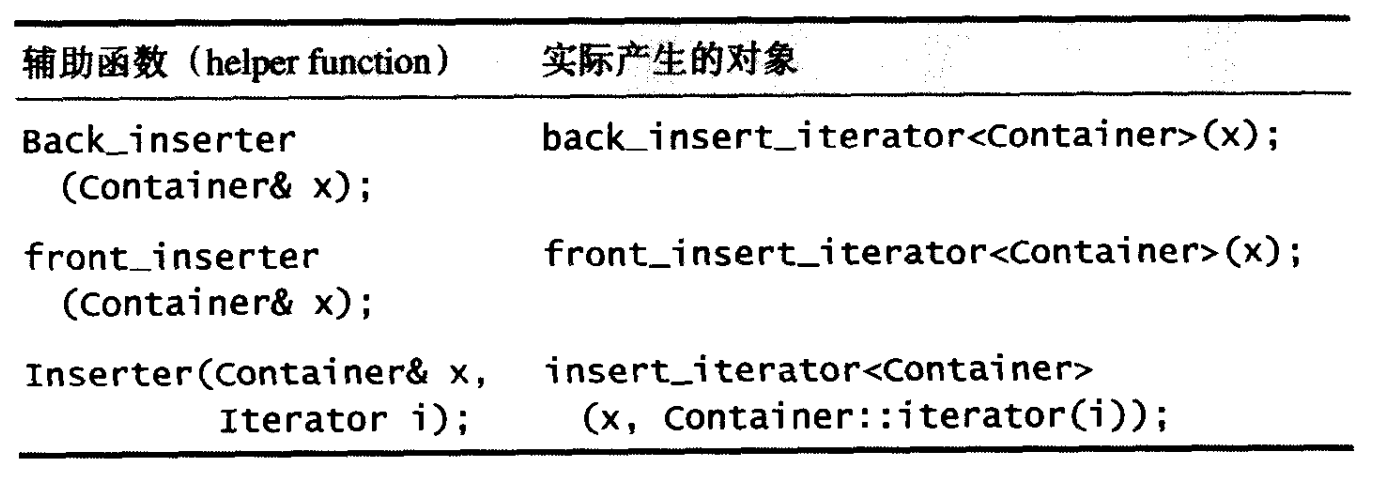
## 配接器运用于迭代器

STL提供了许多应用于迭代器上的适配器，包括insert iterator，reverse iterator，iostream iterator。

1. insert iterator

将一般迭代器的赋值操作转变为插入操作，包括尾端插入的back\_insert\_iterator，头部插入的front\_insert\_iterator和任意位置插入的insert\_iterator。由于这三个的使用接口不太直观，因此提供了三个对应的函数back\_inserter()，front\_inserter()，inserter()来提升使用时的便捷性。

实现观念上，每一个insert iterator内部维护一个容器，当用户对insert iterator进行赋值操作时，被insert iterator内部转换成换插入操作，即对operator=做重载，调用容器底层的push\_front()，push\_back()或insert()函数。其他迭代器的常规操作都是被禁止的，因此将insert\_iterator标记为output iterator。



1. back\_insert\_iterator

类中维护一个容器(指针形式)，为protected属性；五种迭代器属性除了iterator\_category是output\_iterator\_tag外，其他的都是void；显式构造函数将参数提供的容器和内部容器绑定起来；operator=操作符调用容器底层的push\_back()函数，并且返回back\_insert\_iterator本身；为了关闭operator\*，operator++()，operator++(int)的功能，在实现上都是直接返回back\_insert\_iterator本身。辅助模板函数back\_insert接收一个容器作为参数，并返回一个back\_insert\_iterator对象。

1. front\_insert\_iterator

和back\_insert\_iterator完全类似，只是operator=操作符调用容器底层的push\_front()函数。

注意：因为vector没有提供push\_front函数，因此不能使用front\_insert\_iterator。

1. insert\_iterator

和back\_insert\_iterator基本类似，但要多提供一个参数作为插入的位置，内部也要多维护一个迭代器；该参数的类型为container::iterator；在operator=调用容器的insert函数后，还要对传入的插入位置进行更新。

1. reverse iterator

可以将一般迭代器的行进方向进行逆转，对于在尾端上实现的算法很有帮助。

1. reverse iterator

类中维护一个正向迭代器current，为protected属性；五种迭代器的属性都是对应正向迭代器的属性；显式构造函数将参数传入的迭代器和内部维护的迭代器绑定；拷贝构造函数直接将迭代器赋值；base函数返回内部维护的迭代器；operator\*先要对内部迭代器--后再调用内部迭代器的operator\*，返回值为reference；operator->直接调用operator\*，返回值为pointer；operator++和operator--分别调用内部迭代器的operator--和operator++，返回值为自身类型引用；operator++(int)和operator--(int)分别调用operator++和operator--，返回值为非自身类型引用；operator+和operator-分别调用内部迭代器的operator-和operator+，然后利用拷贝构造函数进行返回；operator+=和operator-=直接调用内部迭代器的operator-=和operator+=，返回值为引用；operator[]通过this指针调用reverse iterator的operator+和operator\*，返回值为operator\*的引用:reference。

利用函数模板(为什么不直接在类里面实现???)对operator==，operator!=，operator>，operator>，operator<=，operator>=进行重载，其中只需实现operator==和operator<即可，其他的直接调用对应的来实现；重载operator-计算两个reverse iterator之间的距离；重载operator+来实现形如number+reverse iterator的操作(这里没有对-进行重载时考虑到一般不会写出number-reverse iterator的代码)。

1. iostream iterator

可以将迭代器绑定在某个iostream对象上，绑定在istream对象上的称为istream\_iterator，绑定在ostream对象上的称为ostream\_iterator。

1. istream\_iterator

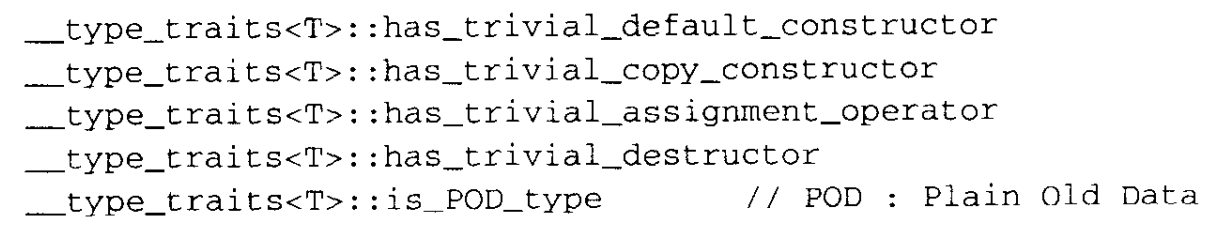
在istream iterator内部维护一个istream对象，对istream iterator所做的++操作被转换为内部istream对象的输入操作(通常为operator>>)；该类模板接受两个参数T和Distance，Distance缺省情况下默认为ptrdiff\_t；protected属性下有一个内部维护的istream对象(指针)，一个类型为T的value，判断输入是否结束的bool型变量end\_marker；用于进行输入的函数read(该函数先判断end\_mark(这里的操作不是很理解???)，在利用内部istream调用operator>>，将值读入到value中，再更新end\_mark的值)；下面是public属性，iterator\_category为input\_iterator\_tag，并且reference和pointer都是const(因为istream iterator的类型为input)；默认构造函数将内部istream对象和cin绑定，end\_mark置为0；构造函数将参数的istream和函数内部istream对象绑定，并直接调用read函数(注意:在声明一个istream对象时要十分小心，不然会一直陷入等待)；operator\*返回value的值；operator->调用operator\*；operator++和operator++(int)都是调用read函数。

1. ostream\_iterator

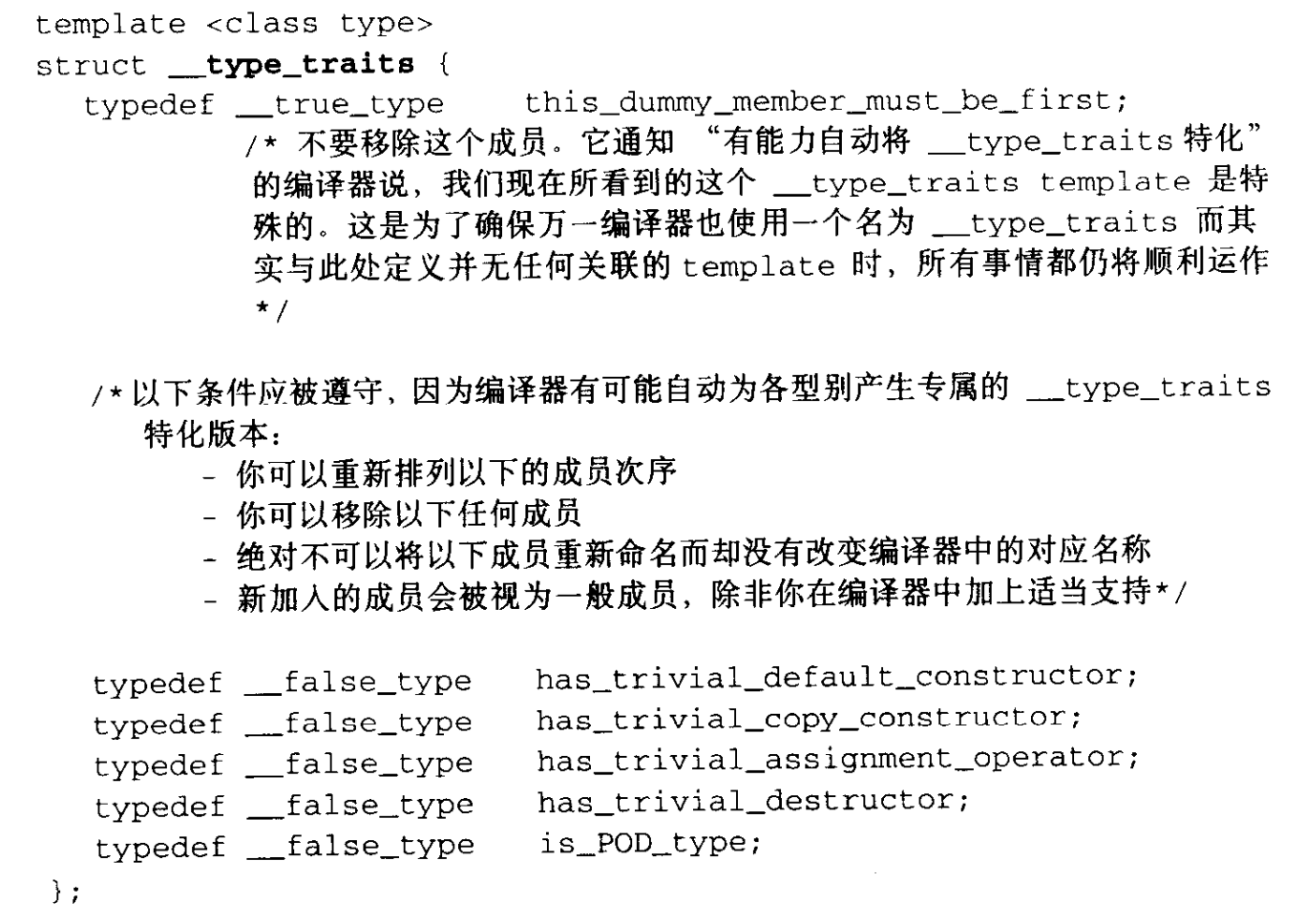
protected属性下维护一个ostream对象和一个const char\*对象(每次输出之后的间隔符)；下面是public属性，五种迭代器属性除了iterator\_category是output\_iterator\_tag外其他的都是void；构造函数有两种，分别是输出指定间隔字符和没有指定输出间隔字符；operator=输出参数提供的数值，并判断间隔符是否为空来输出，返回值为本身；operator\*，operator++和operator++(int)都是直接返回本身。

## Type\_traits

type\_traits负责萃取类型的特性，这里的特性包括：是否具备non\_trivial default ctor，是否具备non\_trivial copy ctor，是否具备non\_trivial assignment operator，是否具备non\_trivial dtor，可以根据这些信息针对具体的类型采取效率最高的措施。

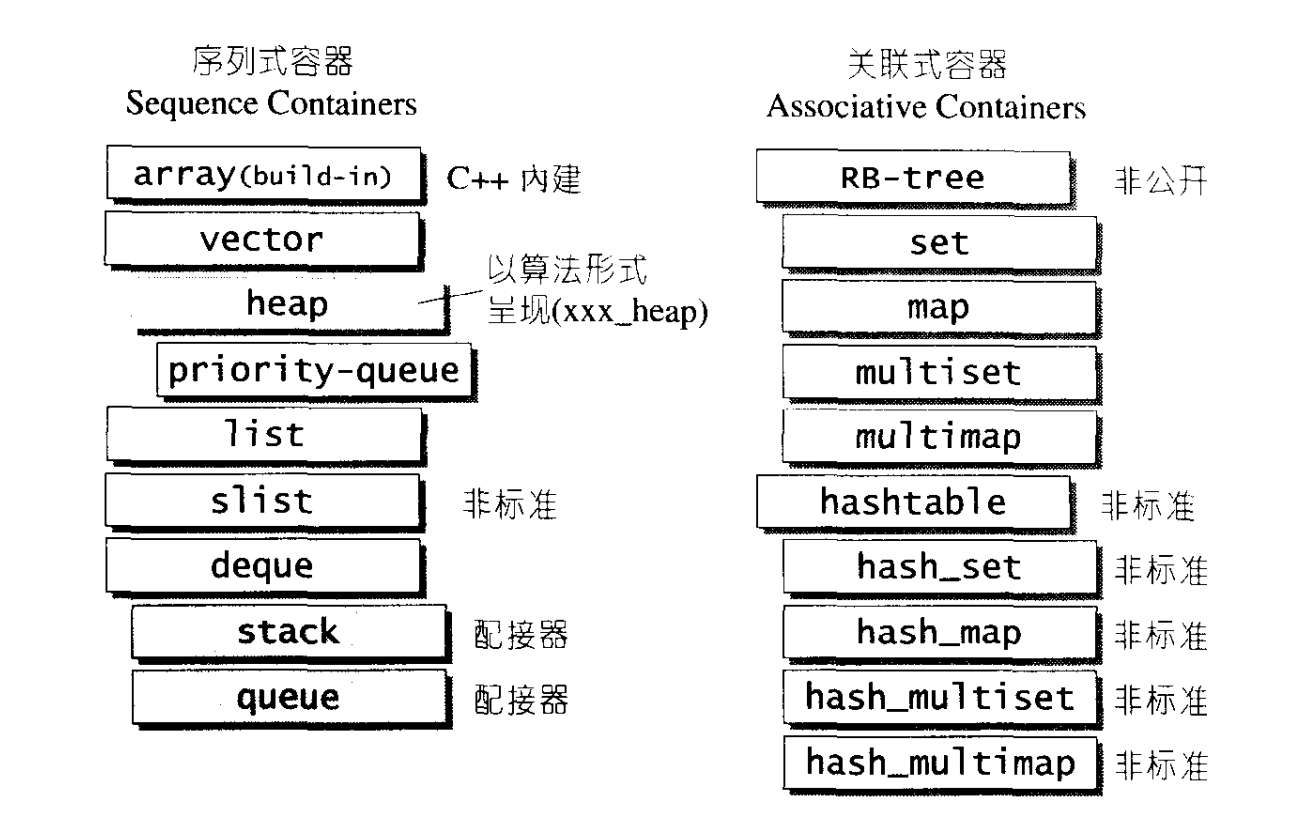


因为我们希望通过这些结果进行参数推导，调用不同的函数，但是true和false都是属于bool类型的，在函数的重载上不能进行区分，因此需要定义两个类来替代true和false，分别为true\_type和false\_type。



对于内置类型准备了全特化版本的type\_tratis，分别有：char, signed char, unsigned char, wchar\_t, short, unsigned short, int, unsigned int, long, unsigned long, long long, unsigned long long, float, double, long double, char\*, signed char\*, unsigned char\*, const char\*, const signed char\*, const unsigned char\*；并且针对一般类型的指针做了偏特化设计。上述所有类型的五种性质都是true\_type类型。

# 3容器



## 序列式容器

C++本身提供了一个序列式容器array，此外STL应该还要提供vector、list、deque、stack、queue和priority\_queue等容器。

### vector

array和vector的区别在于array的静态空间，需要确定要分配的大小后再分配，一旦分配了就不能再改变；vector是动态空间，可以根据元素的加入自行扩充空间。

vector是一个模板类，有两个参数T和Alloc，分别代表元素类型和分配器类型，其中Alloc默认值为alloc。

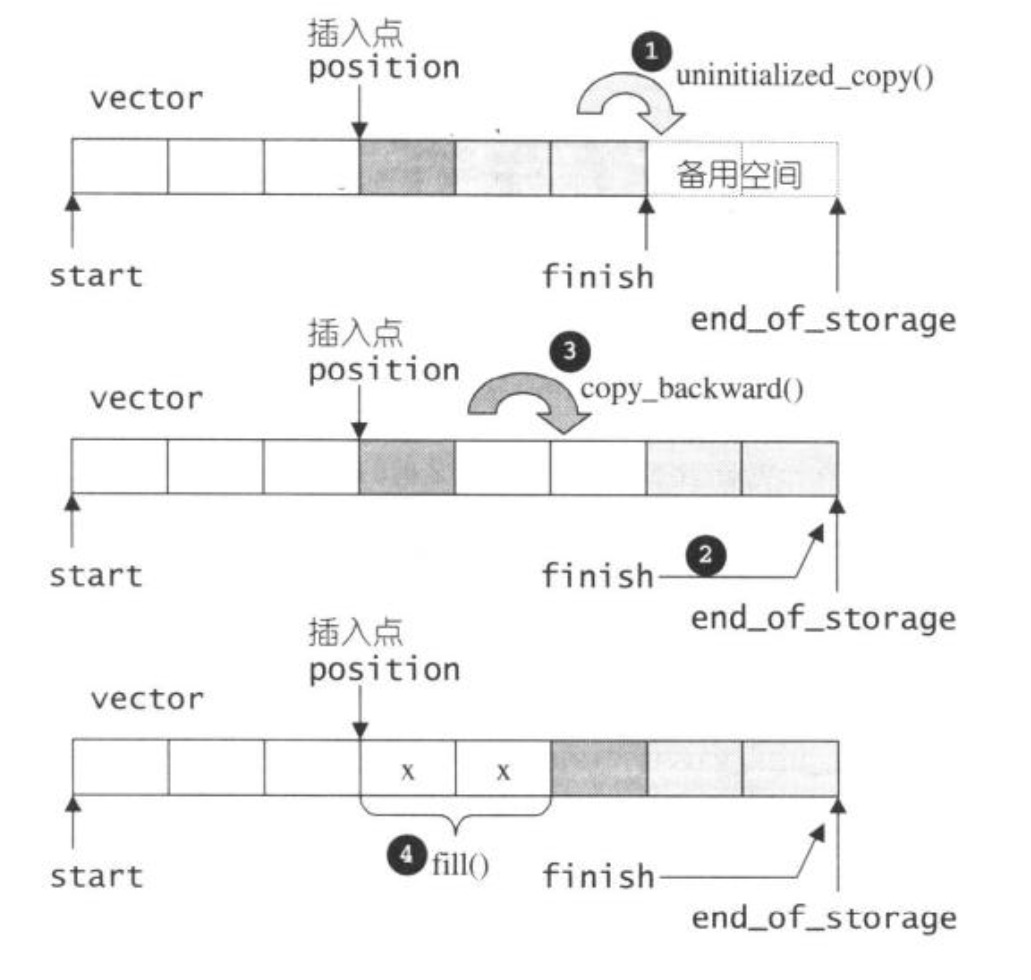
1. vector的嵌套型别定义

属于public属性，分别有

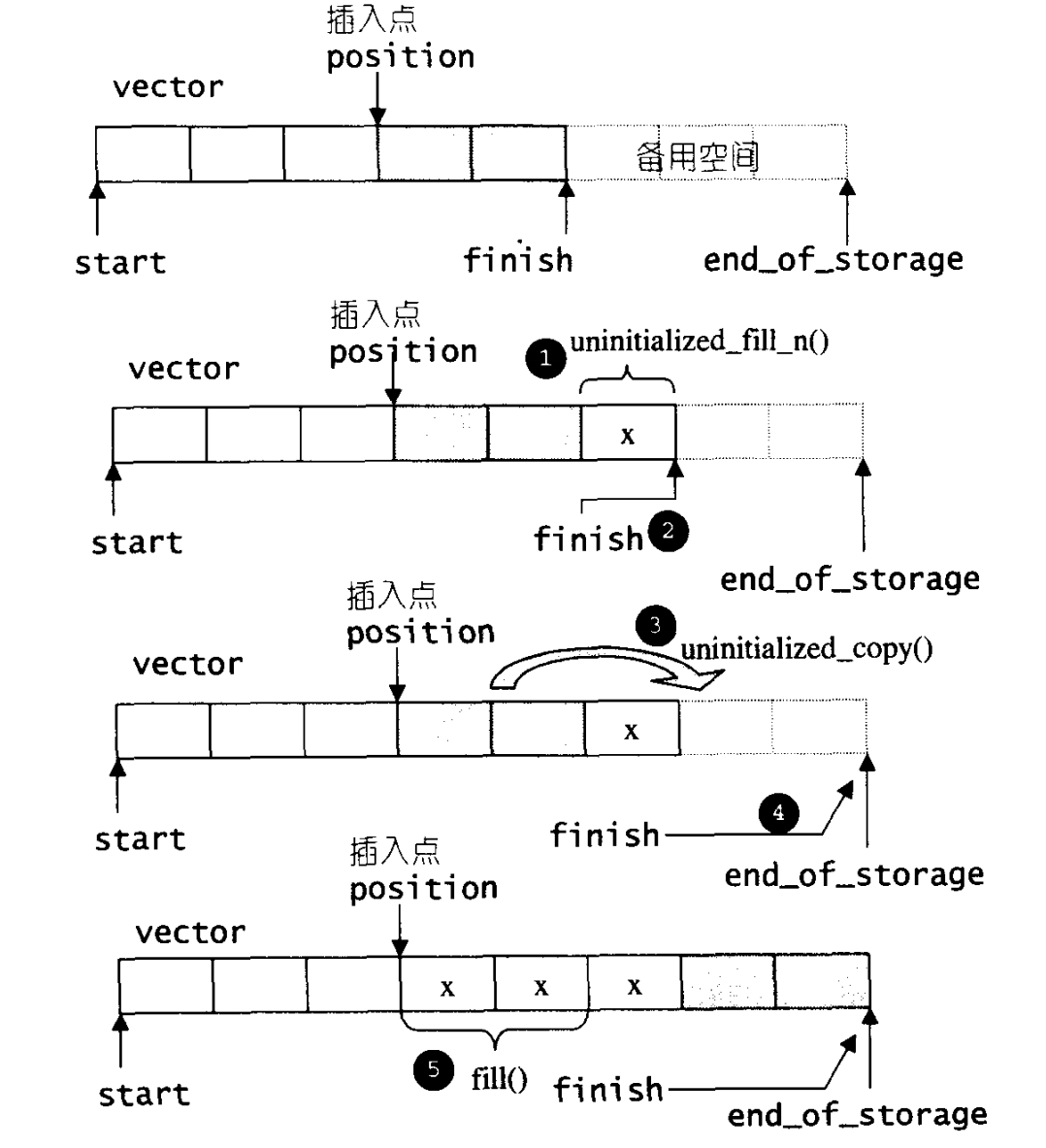
1. data\_allocator(使用的是封装过的符合stl标准的simple\_alloc)
2. allocator\_type(和data\_allocator一样)
3. value\_type
4. pointer
5. const\_pointer
6. iterator
7. const\_iterator
8. reference
9. const\_reference
10. size\_type(size\_t)
11. difference\_type(ptrdiff\_t)
12. reverse\_iterator
13. const\_reverse\_iterator

get\_allocator函数返回一个迭代器。private属性下分别维护三个迭代器start, finish和end\_of\_storage。

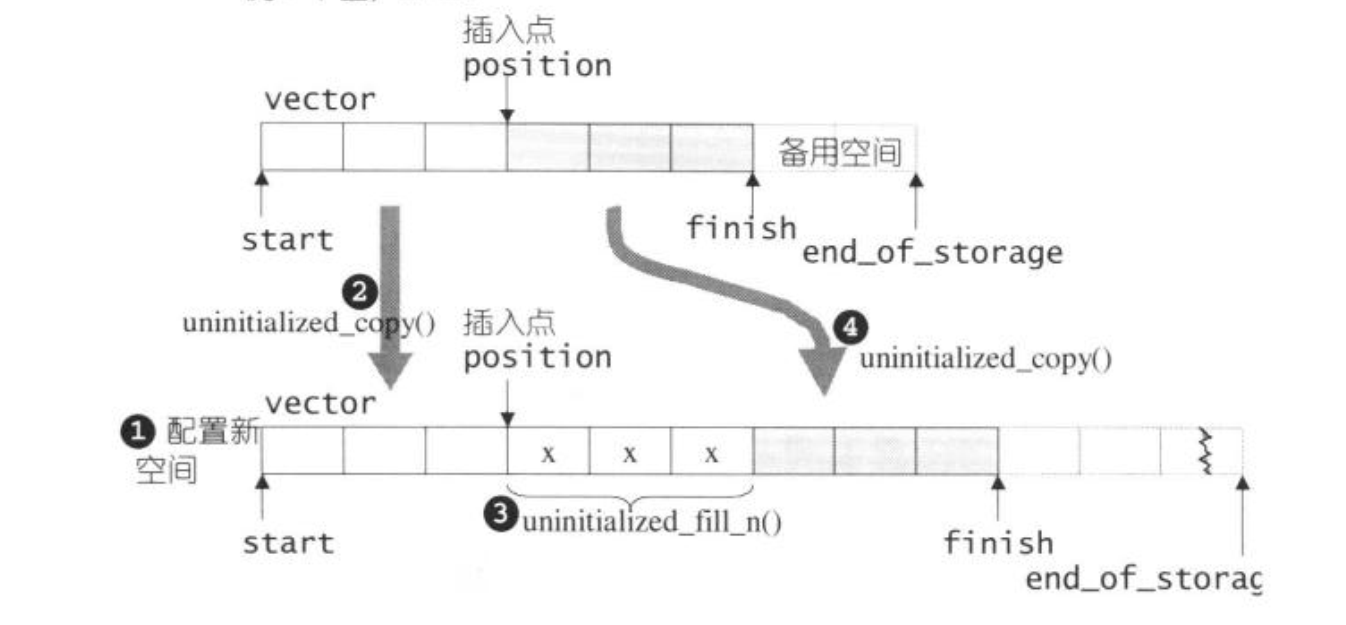
1. 迭代器相关操作
2. begin函数：返回start。
3. begin函数：返回值类型为const\_iterator，是一个const成员函数，也是返回start。
4. end函数：返回finish。
5. end函数：返回值类型为const\_iterator，是一个const成员函数，也是返回finish。
6. rbegin函数：返回反向迭代器类型的finish(end函数获得)。
7. rbegin函数：返回值类型为const的反向迭代器，是一个const成员函数，返回反向迭代器类型的finish(end函数获得)。
8. rend函数：返回反向迭代器类型的start(begin函数获得)。
9. rend函数：返回值类型为const的反向迭代器，是一个const成员函数，返回反向迭代器类型的start(begin函数获得)。
10. cbegin函数：返回const类型的迭代器，是一个const成员函数。
11. cend函数：返回const类型的迭代器，是一个const成员函数。
12. crbegin函数：返回const类型的反向迭代器，是一个const成员函数。
13. crend函数：返回const类型的反向迭代器，是一个const成员函数。
14. 容量相关操作(涉及到的大小数据类型都是size\_type)
15. size函数：返回finish与start的差值。
16. max\_size函数：返回size\_type类型下最多能存放类型为T对象的个数。
17. capacity函数：返回end\_of\_storage与start的差值。
18. empty函数：判断start和finish是否相等。
19. reserve函数：有一个参数n；首先判断n是否大于当前的容量，若为真则可以利用三个函数完成元素搬家操作；首先第一个函数负责向内存空间申请n的大小，然后将元素copy到新的位置；第二个函数负责将原来的元素全部析构掉(注意这里的元素个数)；第三个函数负责将原来容器所占的空间释放掉(注意这里空间的大小)；然后将三个迭代器设为相应的值。
20. 访问元素相关操作
21. operator[]重载函数：有一个参数n，返回值为reference，首先判断n是否越界，返回第n个元素。
22. operator[]重载函数：有一个参数n，返回值为const\_reference，是一个const成员函数；首先判断n是否越界，返回第n个元素。
23. at函数：有一个参数n，返回值为reference，直接调用operator[]。
24. at函数：有一个参数n，返回值为const\_reference，是一个const成员函数，直接调用operator[]。
25. front函数：返回值为reference，返回第一个元素。
26. front函数：返回值为const\_reference，返回第一个元素。
27. back函数：返回值为reference，返回最后个元素。
28. back函数：返回值为const\_reference，返回最后一个元素。
29. data函数：返回值为pointer，返回begin。
30. data函数：返回值为const\_pointer，返回begin。
31. 构造、复制和析构函数
32. 默认构造函数：直接把start，finish和end\_of\_storage全部设为空。
33. 构造函数：有一个参数n，注意要将这个构造函数声明为explicit避免隐式转换；先调用data\_allocator的allocate函数分配内存空间，将返回值赋给start，再调用uninitialized\_fill\_n函数进行初始化uninitialized\_fill\_n的参数分别为start指针、n和初始值，因为没有指明初始值，这里传入临时对象T()，用T对象的默认初始值进行初始化，返回值赋给finish，end\_of\_storage为start指针移动n个位置。
34. 构造函数：有两个参数，分别是个数n和初始值value；操作和前一个构造函数类似，只是uninitialized\_fill\_n函数的参数由默认值改为了value。
35. 复制构造函数：先调用allocator的allocate函数分配rhs.size()个内存空间，并把返回值赋给start，再调用uninitialized\_copy函数将rhs的内容拷贝过来，返回值赋给finish，start移动size个单位就是end\_of\_storage指针。
36. 构造函数：这是一个模板函数，有两个参数first和last；因为传入两个整数(想要使用构造函数3)也可能会导致此模板实例化，但是这个构造函数的参数含义是迭代器的首尾指针，因此需要判断参数类型是否为整数(使用std的is\_integral<>::value)，因为这里的返回值是bool类型，需要转换为true\_type和false\_type类型以进行后面的参数推导(根据这里的值选择要执行的函数)，因此在type\_traits中设计了一个类Is\_integer来进行转换；如果是true\_type则做构造函数3的操作，否则(这里的操作可以单独设计一个函数来完成，因为后面用初始化列表进行构造时的操作和这里一样)先利用distance函数计算出元素个数，再利用allocator的allocate函数进行内存空间的分配并用uninitialized\_copy函数进行初始化。
37. 构造函数：参数是一个初始化列表(std::initializer\_list<value\_type>)，因为可以将初始化列表看做一个容器，因此可以使用构造函数5进行构造。
38. 赋值操作符重载：参数为vector。接受一个参数rhs，首先判断rhs是否等于当前vector，如果不相等，得到rhs的size为len，如果len大于当前vector的容量，则利用allocate\_and\_copy分配新的空间，再设置三个迭代器的值；如果len小于当前vector的size，则将rhs的值copy到当前vector，并将多余的元素destroy；否则将rhs一部分的值copy到当前容量，多余的值通过uninitialized\_copy到当前vector尾端。返回值为当前vector。
39. 赋值操作符重载：参数为初始化列表。利用构造函数构造一个新的vector，再利用swap函数进行交换。
40. 析构函数：分别调用destory和deallocate函数进行元素的析构和内存的释放(可以放在一个函数中完成)，然后将三个迭代器设为空。
41. 修改容器相关操作
42. assign函数：赋值函数，有两个版本，第一个版本接受两个参数，分别是元素个数n和初始值value；实现上调用一个辅助函数fill\_assign完成；第二个版本接受两个参数，分别是first和last，用函数模板实现，因为可能和第一个版本冲突，这里需要判断输入是否为整数(assign\_dispatch)(用Is\_integer来判断)，如果是整数则直接调用fill\_assign辅助函数，否则调用另一个辅助函数assign\_aux。
43. emplace函数
44. emplace\_back函数
45. push\_back函数：接受一个参数值value，如果当前还有容量，则直接利用construct函数并更新finish；如果没有容量，则调用辅助函数insert\_aux。
46. pop\_back函数：检查当前是否有元素，如果有则destroy最后一个元素并调整finish指针。
47. insert函数：接受两个参数position和value；先检查position是否合理；如果当前还有容量并且是在容器最后进行插入，则直接利用construct函数，否则调用insert\_aux函数。返回值为一个迭代器(离start的距离为position-start)。
48. insert函数：插入多个值；这是一个模板函数，接受三个参数，position, first和last；由于需要判断模板参数是整数还是迭代器，分为两个版本(insert\_dispatch)；如果是整数则调用fill\_insert函数；如果是迭代器则调用range\_insert函数(为什么还要一个单独的整数的版本)。
49. erase函数：接受一个参数position，如果删除的不是最后一个元素，则将position之后的元素向前copy一个距离，然后destroy最后一个元素，调整finish指针，返回值为position。
50. erase函数：接受两个参数first和last，将last到finish的元素copy到first的位置得到迭代器i，再将从i到finish的元素全部destroy，调整finish，返回值为first迭代器。
51. clear函数：调用erase函数将start到finish的元素全部删除。
52. resize函数：接受两个参数new\_size和value；如果new\_size比size小，则直接erase掉多余的元素；如果new\_size比size大，则在末尾insert多出的元素，值为value。
53. resize函数：接受一个参数new\_size，调用两个参数的resize函数，第二个参数为T类型的临时变量。
54. 辅助函数
55. allocate\_and\_copy：接受三个参数n，first和last；是一个模板函数，首先allocate n个元素大小空间，然后利用uninitialized\_copy函数将first到last的元素拷贝到新的空间中(这是一个try和catch，如果抛出异常，则将新空间deallocate)。
56. allocate\_and\_fill\_n：思想和allocate\_and\_copy一样。
57. initialize\_aux：模板构造函数调用，根据参数是否是整数调用不同的函数。
58. range\_initialize：是一个模板函数，接受三个参数first，last和tag，根据tag分为两个版本；input\_iterator版本直接将first到last的元素push\_back进容器；forward\_iterator版本先利用distance函数计算出大小，然后allocate出空间，再用uninitialized\_copy函数进行初始化，调整三个迭代器。(在实现上没有区分迭代器的类型，统一使用forward\_iterator的实现)。
59. fill\_insert：接受三个参数，position，n和value；首先判断n是否为0；如果不为0则判断当前剩余容量是否大于n，如果大于等于n则算出position到finish的元素个数elem\_after，如果elem\_after大于n(这里没有等于n是因为后面还要进行copy\_backward)，先将finish处前n个元素通过uninitialized\_copy到finish之后，然后将position之后剩余的元素copy\_backward到finish，再用fill函数在position位置进行填充，调整finish；



如果elem\_after小于等于n，则先在finish处用uninitialized\_fill填充n-elem\_after个value元素，然后调整finish的值，再用uninitialized\_copy将position后的元素copy到finish之后，调整finish的值，再用fill函数在position位置进行填充；



如果剩余容量小于n，则先计算出新空间所需大小，这个大小为old\_size+max(old\_size, n)，然后allocate新的空间，然后用uninitialized\_copy将前半段数据copy到新空间，用uninitialized\_fill函数填充空间，再用uninitialized\_copy函数将后半段数据copy到新空间(这里用try和catch，如果在移动数据过程中抛出了异常，则将新空间destroy和deallocate)，然后将原来的空间destroy和deallocate，调整三个迭代器。



1. range\_insert(input\_iterator版本)：接受四个参数，position，first，last和input\_iterator\_tag；对first到last的每个元素进行insert。
2. range\_insert(forward\_iterator版本)：和fill\_insert函数思想一样，只是其中涉及到Forward\_iterator的情况要用distance和advance函数。
3. fill\_assign函数：辅助函数，接收两个参数n和value；如果n大于当前容器容量，则利用构造函数得到一个新的容器，并利用swap函数进行交换；如果n大于size，则利用fill函数将前size个元素赋值为value，再利用uninitialized\_fill\_n函数将后面的元素赋值；此外，利用fill\_n函数将前n个元素重新赋值，再利用erase函数将后面的元素清除。
4. assign\_aux函数：是一个模板函数，有两个版本，第一个版本针对input\_iterator，因为input\_iterator不能写，因此声明一个vector的iterator cur辅助写，如果first先到达终点，则将cur之后的元素都erase；如果cur先到达终点，则利用insert函数在cur之后进行插入；第二个版本针对forward\_iterator，先利用distance函数计算last和first之间的距离len，如果len大于当前容量，则重新分配空间并将原来空间销毁掉；如果size大于len，利用copy函数将元素拷贝到vector中，然后利用将剩下的元素destory；此外，将待处理元素分为两部分，第一部分用copy函数赋值到vector中，第二部分用uninitialized\_copy处理。
5. insert\_aux函数：接受两个参数position和value；首先判断当前是否还有容量，如果有则在finish位置上构造一个新的元素，值和前一个元素相同，然后调用copy\_backward函数将position以后的元素都向后移动一个距离，再将position位置上的值设为value；如果没有容量，则根据之前的容量进行扩容，如果之前容量是0则扩容为2，否则进行二倍扩容，再分配新的空间，分别用uninitialized\_copy和construct函数将元素拷贝到新的空间中(这里要用到try和catch，如果抛出异常则进行destroy和deallocate)；然后将原来空间destroy并deallocate，设置三个指针。
6. swap函数：如果参数和当前容器不相等，则利用swap交换三个迭代器。
7. 常用函数(以下的都不再属于vector类中)
8. operator==：接受两个参数lhs和rhs，比较两个容器的size，并利用equal函数对元素进行比较。
9. operator<：接受两个参数lhs和rhs，利用lexicographical\_compare函数对两个容器的元素进行比较。
10. swap函数：接受两个参数lhs和rhs，直接利用lhs调用成员函数swap。
11. operator!=：调用operator==。
12. operator>：调用operator<。
13. operator<=：调用operator>。
14. operator>=：调用opeartor<。

### List

1. list的节点结构

list的节点有两部分，list\_node\_base和lise\_node，两部分都是模板类。

1. list\_node\_base

包含两个typedef，分别定义base\_ptr和node\_ptr；有两个base\_ptr类型的指针prev和next；self函数返回自身(需要转型)；unlink函数将两个指针都指向自身；as\_node函数返回一个node\_ptr类型的指针(将自身转型)；构造函数为默认。

1. list\_node

继承list\_node\_base，包含一个数据data，有一个默认构造函数，接受一个参数的构造函数，同样包含self函数和as\_base函数。

1. list的迭代器

list的迭代器包含两部分list\_iterator\_base和list\_iterator。

1. list\_iterator\_base

继承迭代器模板mystl::iterator，迭代器类型为bidirectional\_iterator，typedef包含size\_type，difference\_type，iterator\_category；内部维护一个base\_ptr指针node；有一个默认构造函数，接受一个参数为base\_ptr的构造函数；有一个自增函数incr和自减函数decr；重载==和!=运算符。

1. list\_iterator

继承list\_iterator\_base，typedef包含value\_type，pointer，reference，base\_ptr，node\_ptr和self(typedef不会继承)；包含一个默认构造函数，一个接受base\_ptr参数的构造函数，一个接受node\_ptr参数的构造函数，一个复制构造函数；重载operator\*函数，先调用as\_node函数然后再访问data，返回值为reference；重载operator->函数，调用operator\*函数，返回值为pointer；重载operator++()，operator++(int)，operator--()和operator--(int)，返回值都是self，前增和前减的返回值为self 的引用。

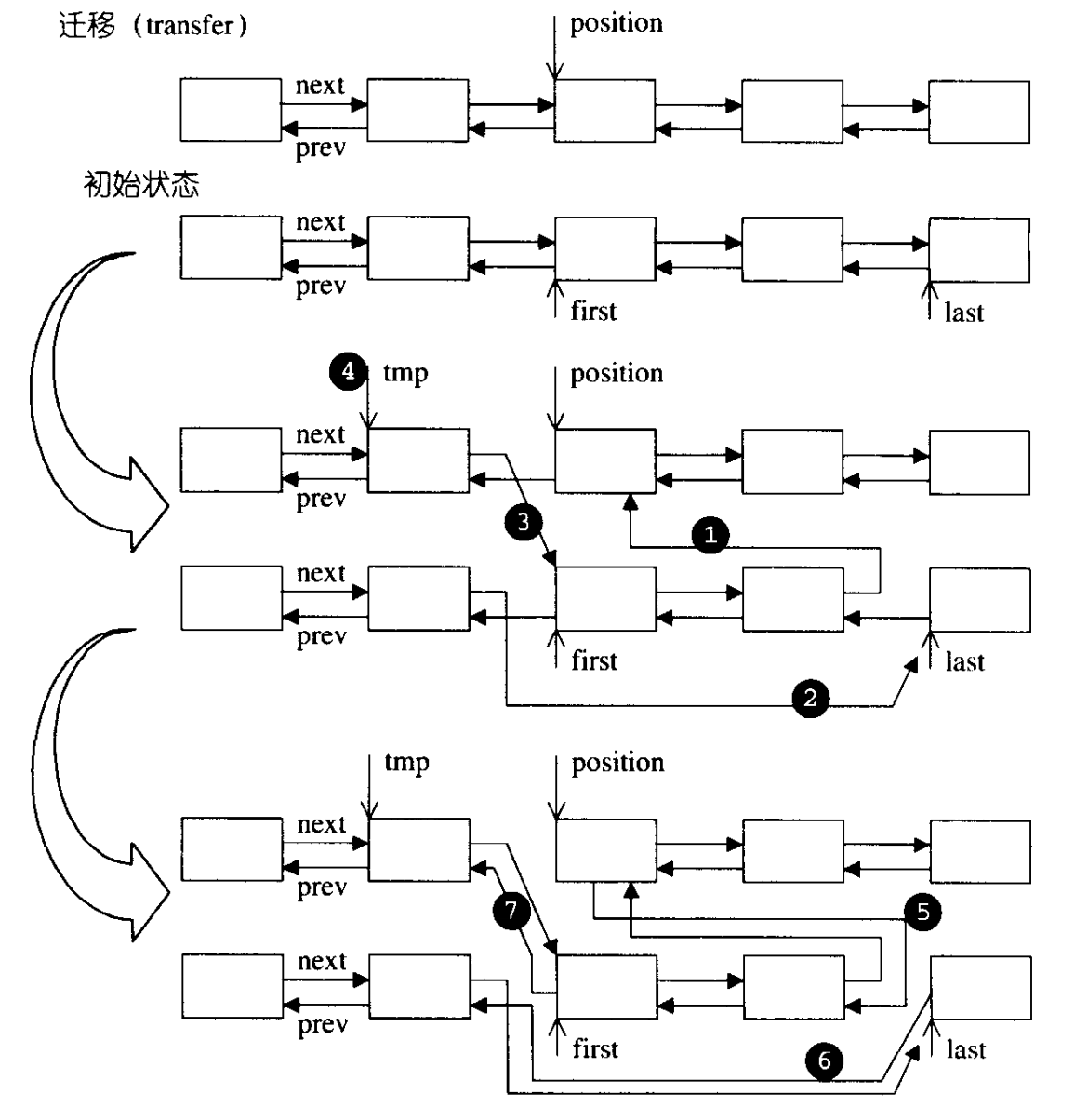
1. list\_const\_iterator

和list\_iterator类似，只是在typedef的pointer和reference，构造函数有所不同。

1. list
2. 基本内容

模板参数为T和Alloc=alloc；两个typedef包括allocator\_type和data\_allocator，都为simple<T, Alloc>；两个typedef包括base\_allocator和node\_allocator；typedef包括value\_type，pointer(value\_type\*)，const\_pointer，reference(value\_type&)，const\_reference，size\_type，difference\_type，iterator，const\_iterator，reverse\_iterator，const\_reverse\_iterator，base\_ptr和node\_ptr；get\_allocator函数返回一个配置器；private属性下维护node\_ptr类型的节点node。

1. 构造、复制和析构函数
2. 默认构造函数：使用node\_allocator的allocate函数分配一个空间，并将返回值赋值给node，再使用unlink函数将两个指针指向自身。
3. 构造函数：声明explicit，接受一个参数n；首先使用node\_allocator的allocate函数分配一个空间，并将返回值赋值给node，再使用unlink函数将两个指针指向自身，然后调用insert函数insert(begin(), n, T())。
4. 构造函数：接受两个参数n和value；首先调用辅助函数construct\_init，然后调用insert函数insert(begin(), n, value)。
5. 构造函数：是一个函数模板，模板参数为InputIterator；首先使用node\_allocator的allocate函数分配一个空间，并将返回值赋值给node，再使用unlink函数将两个指针指向自身，然后调用insert函数insert(begin(), first, last)。这里不需要根据迭代器的类型做不同操作，因为这些在insert中会实现。
6. 构造函数：接受一个std::initializer\_list参数，实现上和上述构造函数相同。
7. 构造函数：接受两个参数first和last，类型都为const T\*，实现上和上述构造函数相同。
8. 构造函数：接受两个参数first和last，类型都为const\_iterator，实现上和上述构造函数相同。
9. 复制构造函数：接受一个参数list<T, Alloc>，实现上和上述构造函数相同。
10. operator=：接受一个参数rhs；首先判断rhs不等于this，然后声明两个iterator分别指向begin()和end()，两个const\_iterator分别指向rhs的begin()和end()；将first2的值赋给first1，如果first2等于last2，就调用erase函数将first1到last1的元素清除，如果first1等于last1，就调用insert函数在last1后面插入first2到last2的元素。返回值为list的引用。
11. operator=：接受一个参数rhs，类型为std::initializer\_list，实现上和上述相同。
12. 析构函数：如果node不为空，先调用clear函数，然后调用data\_allocator的deallocate函数将node销毁，然后将指针设为空。
13. 迭代器相关操作
14. begin函数：返回头结点，即node->next，返回值类型为iterator。
15. begin函数：const函数，返回值类型为const\_iterator，返回node->next。
16. end函数：返回尾结点，即node，返回值类型为iterator。
17. end函数：const函数，返回值类型为const\_iterator，返回node。
18. rbegin函数：返回值类型为reverse\_iterator，返回end()的reverse\_iterator。
19. rbegin函数：const函数，返回值类型为const\_reverse\_iterator，返回end()的reverse\_iterator。
20. rend函数：返回值类型为reverse\_iterator，返回begin()的reverse\_iterator。
21. rend函数：const函数，返回值类型为const\_reverse\_iterator，返回begin()的reverse\_iterator。
22. cbegin函数：const函数，返回值类型为const\_iterator，返回begin()。
23. cend函数：const函数，返回值类型为const\_iterator，返回end()。
24. crbegin函数：const函数，返回值类型为const\_reverse\_iterator，返回rbegin()。
25. crend函数：const函数，返回值类型为const\_reverse\_iterator，返回rend()。
26. 容量相关操作
27. empty函数：判断node->next是否等于node。
28. size函数：利用distance函数计算begin到end的距离并返回。
29. max\_size函数：返回size\_type的-1。因为链表理论上是无限大小的。
30. 访问元素相关操作
31. front函数：返回值类型为reference，断言链表非空，返回第一个元素。
32. front函数：const函数，返回值类型为const reference，断言链表非空，返回第一个元素。
33. back函数：返回值类型为reference，断言链表非空，返回最后一个元素。
34. back函数：const函数，返回值类型为const reference，断言链表非空，返回最后一个元素。
35. 调整容器相关操作
36. assign函数：接受两个参数n和value；实现上调用辅助函数fill\_assign。
37. assign函数：是一个函数模板，模板参数为InputIterator，参数为first和last；利用Is\_integer判断模板参数是否为整数，并调用assign\_dispatch辅助函数。
38. assign函数：接受一个std::initializer\_list参数；实现上依次给list赋值，多出元素就insert，少于就erase。
39. insert函数：接受两个参数position和value；首先断言当前size+1小于等于max\_size，然后调用辅助函数create\_node创建一个结点，然后设置四个指针的值，返回值类型为iterator，返回创建的新结点。
40. insert函数：接受一个参数position；实现上调用接受两个参数的insert函数，第二个参数值为T的默认值T()。
41. insert函数：是一个函数模板，模板参数为InputIterator，参数为position，first和last，实现上根据Is\_integer判断模板参数是否为整数，调用辅助函数insert\_dispatch。
42. insert函数：接受三个参数position，n和value；调用fill\_insert函数。
43. insert函数：接受三个参数position，first和last，其中first和last都是const T\*类型；实现上利用for循环一直在position处进行插入。
44. insert函数：接受三个参数position，first和last，其中first和last都是const iterator类型；实现上利用for循环一直在position处进行插入。
45. push\_front函数：调用insert函数在begin处进行插入。
46. push\_front函数：接受一个参数value；调用insert函数在begin处进行插入。
47. push\_back函数：调用insert函数在end处进行插入。
48. push\_back函数：接受一个参数value；调用insert函数在end处进行插入。
49. pop\_front函数：调用erase函数清除begin结点。
50. pop\_back函数：调用erase函数最后一个结点的值(end的前一个结点)。
51. erase函数：接受一个参数position；修改四个指针的值，利用destroy函数将结点数据域的值清除，再调用put\_node函数销毁结点，返回值类型为iterator，返回position的下一个结点。
52. erase函数：接受两个参数first和last；实现上利用for循环调用erase函数清除first迭代器的结点，返回值类型为iterator，返回last。
53. clear函数：利用while循环对所有的结点进行destroy和put\_node的操作，最后把头结点的两个指针指向自身。注意操作时结点的类型。
54. resize函数：接受两个参数new\_size和value；实现上声明一个临时迭代器i和len；利用for循环进行空遍历，如果len等于new\_size，利用erase清除i到end之间的元素；如果i等于end，利用insert函数在end后插入new\_size-len个新元素。
55. resize函数：接受一个参数new\_size；通过调用两个参数的resize函数实现，第二个参数为T的默认值T()。
56. swap函数：接受一个参数rhs；利用mystl的swap函数交换两个node。
57. list相关操作
58. splice函数：接受两个参数position和other(list)；如果other非空，则利用transfer函数将other.begin()到other.end()移动到position前。
59. splice函数：接受三个参数position，other(list)和i；声明一个临时迭代器j为i的下一个迭代器，如果position为i或者j(即要插入的结点是position或者position前一个结点)，直接return；否则调用transfer函数。(这里的待插入结点可能来自当前list也可能来自其他list)。
60. splice函数：接受四个参数position，other(list)，first和last；如果first不等于last，就直接调用transfer函数。(这里的待插入结点可能来自当前list也可能来自其他list)。
61. remove函数：接受一个参数value；利用while循环，如果遍历到当前结点的值等于value则调用erase函数清除当前结点。
62. unique函数：清除两个相邻的相同结点；如果list为空则直接返回，利用while循环，如果first等于next则利用erase函数清除next，否则first等于next。
63. merge函数：将两个list合并为一个有序的list(两个list都是有序的)，接受一个参数other(list)；利用while循环，如果first2的值小，就利用transfer函数进行插入，否则++first1，如果while循环后first2不等于last2，利用transfer函数将first2到last2的结点插入到last1后。
64. reverse函数：如果list的大小是0或1则直接return，利用while循环将每个结点利用transfer函数插入到begin处。
65. sort函数：类似于归并排序和二进制加法(每次抽出一个元素插入到当前运算器中，并实时记录当前最高层数)；当list的元素个数是0或者1时直接返回，用到一个辅助list carry(carry负责取出原始链表的头一个数据节点和交换数据中转站作用)和一个辅助list数组counter[64](counter负责存储排序后的部分list，counter[i]里面最多存储数据个数为2^(i+1))，fill变量表示当前可处理数据的个数为2^(fill)；首先利用while循环判断当前list是否为空，空就退出循环，然后将当前list的第一个元素splice给carry，定义一个变量i表示当前处理的层数，while中如果i小于fill并且当前层数的counter中有元素，就将carry中的元素merge进当前层carry中，然后把当前层的元素swap到carry中再将当前处理层数++，跳出当前while循环后将carry的元素swap到当前处理层数的counter中，然后判断当前处理层数是否到达了最高层数fill，如果是则将最高层数fill++；最后将所有的counter(0到fill)依次merge，最后将最高层的counter swap到当前list中。
66. remove\_if函数：是一个模板函数，模板参数为Predicate，接受一个参数Pred；实现上和remove函数类似，只是在每次erase元素前要先判断是否满足Pred的条件。
67. unique函数：是一个模板函数，模板参数为BinaryPredicate，接受一个模板参数binary\_pred；实现上和unique函数类似，把两个相邻元素相同的条件换为了两个相邻元素满足binary\_pred的条件。
68. merge函数：是一个模板函数，模板参数为StrictWeakOrdering，接受两个参数other(list)和comp；实现上和merge函数类似，在比较大小时换成了comp(\*first2, \*first1)。
69. sort函数：是一个模板函数，模板参数为StrictWeakOrdering，接受一个参数comp；实现上和sort函数类似，在调用merge函数时调用的含有两个参数的merge函数，第二个参数为comp。
70. 辅助函数
71. allocate\_and\_init函数：分配一个node空间给node，并把prev和next指针指向自身。
72. fill\_assign函数：接受两个参数n和value；声明一个iterator等于begin，同时++i和--n，并将value的值赋给i，如果n大于0，利用insert函数在end后插入n个value元素；否则利用erase函数删除i到end的元素。
73. assign\_dispatch函数：模板函数，模板参数为Integer，接受三个参数n，value和true\_type；实现上调用fill\_assign函数。
74. assign\_dispatch函数：模板函数，模板参数为InputIterator，接受三个参数first，last和false\_type；声明两个iterator分别指向begin和end，first1和first2同时++，并将first2的值赋给first1，如果first1遍历结束，调用insert函数在end后插入first2到end2的元素；否则调用erase函数删除first1到end1之间的元素。
75. create\_node函数：首先调用辅助函数get\_node得到一个结点，然后利用data\_allocator的construct函数赋初值，这里用try和catch实现，catch里面调用辅助函数put\_node，返回值为node\*，返回新创建的结点。
76. create\_node函数：接受一个参数value；实现上和d相同，只是在construct时的初值不同。
77. get\_node函数：利用node\_allocator的allocate函数创建一个结点，返回值类型为List\_node<T>\*，返回allocate的返回值。
78. put\_node函数：接受一个参数结点p，利用node\_allocator的deallocate函数进行结点的销毁。
79. insert\_dispatch函数：模板函数，模板参数为Integer，接受四个参数pos，n，value和true\_type；实现上调用辅助函数fill\_insert实现。
80. insert\_dispatch函数：模板函数，模板参数为Integer，接受四个参数pos，n，value和false\_type；实现上利用for循环一直在position处调用insert函数进行插入。
81. fill\_insert函数：接受三个参数position，n和value；实现上利用for循环一直在position处进行插入。
82. transfer函数：将first到last的序列插入到position之前，接受三个参数position，first和last；首先判断position不是last，然后修改6个指针的值(3<- ->，3<-)，实现顺序如下图所示。



1. 常用函数
2. operator==：接受两个参数lhs和rhs，类型都是const list；声明四个const iterator，分别进行比较，最后判断两个list是否都遍历结束。
3. operator<：接受两个参数lhs和rhs，类型都是const list；利用mystl的lexicographical\_compare进行比较。
4. operator!=：接受两个参数lhs和rhs，类型都是const list；调用operator==。
5. operator>：接受两个参数lhs和rhs，类型都是const list；调用operator<。
6. operator<=：接受两个参数lhs和rhs，类型都是const list；调用operator<。
7. operator>=：接受两个参数lhs和rhs，类型都是const list；调用operator<。
8. swap函数：接受两个参数lhs和rhs，类型都是list；利用lhs的成员函数swap实现。

### deque

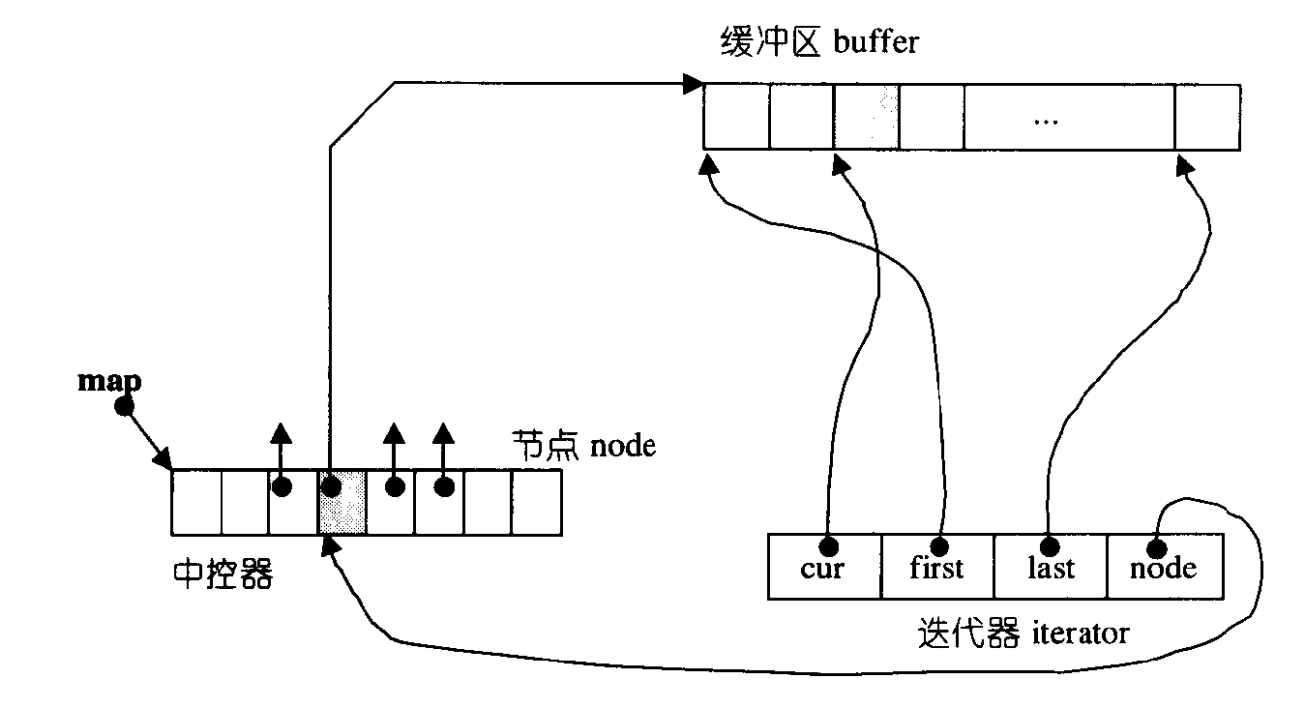
1. buf大小

定义了一个模板类deque\_buf\_size，模板参数为T，有一个静态成员变量value(size\_t)(这里可以用constexpr)，如果sizeof(T)小于512则value等于512/sizeof(T)，否则value等于1。

1. deque的迭代器

是一个模板类，有三个模板参数T，Ref和Ptr，public继承iterator模板，iterator类型为random\_access\_iteartor。

1. typedef包含了iterator和const\_iterator(其中三个模板参数都用T表示)。
2. typedef包含了value\_type，pointer，reference，size\_type，difference\_type，self(deque\_iterator)和map\_pointer(T\*\*)。
3. 有一个const静态成员变量buffer\_size等于deque\_buf\_size中value的值。
4. 包含的成员变量有：cur(T\* 指向所在缓冲区的当前元素)，first(T\* 指向当前缓冲区的第一个元素)，last(T\* 指向当前缓冲区的最后一个元素)和node(map\_pointer 指向缓冲区所在的结点)。



1. 默认构造函数：将四个指针全部设为空。
2. 构造函数：接受两个参数v(T\*)和n(map\_pointer)，将cur的值设为v，first的值设为\*n，last的值设为\*n+buffer\_size，node的值设为n。
3. 复制构造函数：接受一个参数rhs(iterator)，将rhs对应的值赋给四个指针。
4. 复制构造函数：接受一个参数rhs(const\_iterator)，将rhs对应的值赋给四个指针。
5. 重载operator=：接受一个参数rhs，如果rhs不等于this，将四个指针设为rhs对应的值；返回值类型为self&，返回自身。
6. 重载operator\*：const成员函数，返回值类型为reference，返回cur的值。
7. 重载operator->：const成员函数，返回值类型为pointer，返回cur。
8. 重载operator-：const成员函数，接受一个参数other(self&)，返回两个迭代器之间的距离，距离为两个迭代器之间的结点个数乘以buf\_size，加上当前迭代器cur到first之间的距离，加上参数迭代器last到cur之间的距离，返回值类型为difference\_type，注意类型转换。
9. set\_node函数：接受一个参数new\_node(map\_pointer)，将node设为new\_node，first设为\*new\_node，last设为first+buf\_size。(注意此处不会设置cur的值)
10. 重载operator++()：对cur进行++，如果cur等于last，调用set\_node函数将node设为node+1，再把cur设为first；返回值类型为self&，返回自身。
11. 重载operator++(int)：先保存this，再对this进行++，返回值类型为self，返回temp。
12. 重载operator--()：如果cur等会first，调用set\_node将node设为node-1，再将cur设为last，将cur--；返回值类型为self&，返回自身。
13. 重载operator--(int)：先保存this，再对this进行--，返回值类型为self，返回temp。
14. 重载operator+=()：接受一个参数n(difference\_type)，计算出最后的偏移量offset(n+cur-first)，如果offset落在当前缓冲区内，则直接对cur进行+=n操作；计算出node偏移量offset\_node，如果offset大于0，offset\_node就等于offset/buf\_size，如果offset小于0，offset\_node就等于-(-offset-1)/buf\_size – 1；调用set\_node函数将设置缓冲区，再调整cur的值，cur加上offset-(buf\_size)\*offset\_node；返回值类型为self&，返回自身。
15. 重载operator+：const成员函数，接受一个参数n(difference\_type)；声明一个temp等于this，然后调用+=并返回，返回值类型为self。
16. 重载operator-=：接受一个参数n(difference\_type)；调用operator+=，返回值类型为self&。
17. 重载operator-：const成员函数，接受一个参数n(difference\_type)；声明一个temp等于this，然后调用-=并返回，返回值类型为self。
18. 重载operator[]：const成员函数，接受一个参数n(difference\_type)；通过this调用operator+后再调用operator\*，返回值类型为reference。
19. 重载operator==：const成员函数，接受一个参数rhs；比较cur和rhs的cur。
20. 重载operator!=：const成员函数，接受一个参数rhs；调用operator==。
21. 重载operator<：const成员函数，接受一个参数rhs；如果node相等就比较cur，否则比较node。
22. 重载operator>：const成员函数，接受一个参数rhs；调用operator<。
23. 重载operator<=：const成员函数，接受一个参数rhs；调用operator<。
24. 重载operator>=：const成员函数，接受一个参数rhs；调用operator<。
25. deque

函数模板，模板参数有两个T和Alloc(默认为alloc)；

1. typedef
2. allocator\_type：声明为simple\_alloc<T, Alloc>。
3. data\_allocator：声明为simple\_alloc<T, Alloc>。
4. map\_allocator：声明为simple\_alloc<T\*, Alloc>。
5. value\_type
6. pointer(value\_type\*)。
7. reference(value\_type&)。
8. const\_pointer。
9. const\_reference。
10. size\_type
11. difference\_type
12. map\_pointer(pointer\*)
13. const\_map\_pointer
14. iterator(deque\_iterator<T, T&, T\*>)
15. const\_iterator
16. const\_reverse\_iterator
17. reverse\_iterator

有一个get\_allocator函数，返回一个配置器；有一个静态const成员变量buffer\_size等于deque\_buf\_size的value，有一个const int变量initial\_map\_size等于8。private属性下维护四个成员变量，map(map\_pointer 指向一个map，map中的每个元素都是一个指针，指向一个缓冲区)，map\_size(size\_t map内指针的数量，也是缓冲区的个数)，start(iterator 指向第一个结点)，finish(iterator 指向最后一个结点的下一个位置)。

1. 构造、复制和析构函数
2. 默认构造函数：将map设为空，map\_size设为0，调用辅助函数initialize\_map(参数为0)初始化map。
3. 构造函数：接受一个参数n(size\_type)，显式声明；先调用辅助函数initialize\_map初始化map，然后调用辅助函数fill\_initialize(T())进行赋值。
4. 构造函数：接受两个参数n和value；先调用辅助函数initialize\_map初始化map，然后调用辅助函数fill\_initialize(value)进行赋值。
5. 复制构造函数：接受一个参数rhs；先调用辅助函数initialize\_map初始化map，然后调用uninitialized\_copy函数进行赋值。
6. 构造函数：接受一个参数std::initializer\_list；实现上和复制构造函数相同，元素个数需要用distance得到。
7. 构造函数：是一个模板函数，模板参数为InputIterator，接受两个参数first和last；利用Is\_integer判断模板参数是否为整数，调用initialize\_dispatch函数。
8. 构造函数：接受两个参数first和last，都是value\_type\*类型；先调用initialize\_map初始化map，然后调用uninitialized\_copy进行赋值。
9. 构造函数：接受两个参数first和last，都是const\_iterator类型；先调用initialize\_map初始化map，然后调用uninitialized\_copy进行赋值。
10. 析构函数：调用destroy函数对所有元素进行清除，判断map是否为空，如果不为空，先调用destroy\_nodes函数(第二参数是finish后一个结点)，然后调用map\_allocator的deallocate函数，将map设为空。
11. operator=函数：接受一个参数rhs；首先判断rhs是否等于当前deque，如果等于则直接返回自身；如果当前deque的长度大于rhs的长度，则先调用copy函数将rhs的元素拷贝到当前deque中，然后调用erase函数将剩余的元素清除；如果当前deque的长度小于等于rhs的长度，先把rhs的部分元素拷贝到当前deque中，然后将剩余的元素通过insert函数插入到当前deque中；返回值类型为deque&，返回自身。
12. operator=函数：接受一个参数rhs(std::initializer\_list)；利用构造函数声明一个临时deque，然后调用swap函数将临时deque和当前deque交换，返回值类型为deque&，返回自身。
13. 迭代器相关操作
14. begin函数：返回start。
15. begin函数：const成员函数，返回值类型为const\_iterator，返回start。
16. end函数：返回finish。
17. end函数：const成员函数，返回值类型为const\_iterator，返回finish。
18. rbegin函数：返回end()的reverse\_iterator。
19. rbegin函数：const成员函数，返回值类型为const\_reverse\_iterator，返回end()的reverse\_iterator。
20. rend函数：返回begin()的reverse\_iterator。
21. rend函数：const成员函数，返回值类型为const\_reverse\_iterator，返回begin()的reverse\_iterator。
22. cbegin函数：const成员函数，返回值类型为const\_iterator，返回begin()。
23. cend函数：const成员函数，返回值类型为const\_iterator，返回end()。
24. crbegin函数：const成员函数，返回值类型为const\_reverse\_iterator，返回rbegin()。
25. crend函数：const成员函数，返回值类型为const\_reverse\_iterator，返回rend()。
26. 容量相关操作：
27. empty函数：const成员函数，判断start和finish是否相等。
28. size函数：const成员函数，返回值类型为size\_type，返回finish-start。
29. max\_size函数：const成员函数，返回值类型为size\_type，返回-1的size\_type(因为deque可以无限扩展)。
30. resize函数：接受两个参数，new\_size和value；如果new\_size小于当前deque长度，调用erase函数清除多余的元素，如果new\_size大于等于当前deque长度，利用insert函数插入多出的元素。
31. resize函数：接受一个参数new\_size，调用两个参数的resize函数，第二参数为T的默认值T()。
32. 访问元素相关操作：
33. operator[]：接受一个参数n；断言n小于size，返回值类型为reference，通过start迭代器调用operator[]并返回，注意类型的转换。
34. operator[]：const成员函数，接受一个参数n；断言n小于size，返回值类型为const\_reference，通过start迭代器调用operator[]并返回，注意类型的转换。
35. at函数：接受一个参数n；断言n小于size，返回值类型为reference，通过this调用operator[]并返回。
36. at函数：const成员函数，接受一个参数n；断言n小于size，返回值类型为const\_reference，通过this调用operator[]并返回。
37. front函数：返回值类型为reference，返回start的值。
38. front函数：const成员函数，返回值类型为const\_reference，返回start的值。
39. back函数：返回值类型为reference，返回finish前一个迭代器的值(注意不要改变finish的值)。
40. back函数：const成员函数，返回值类型为const\_reference，返回finish前一个迭代器的值(注意不要改变finish的值)。
41. 修改容器相关操作：
42. assign函数：接受两个参数n和value；实现上调用辅助函数fill\_assign。
43. assign函数：模板函数，模板参数为InputIterator，接受两个参数first和last；利用Is\_integer判断模板参数是否是整数，然后调用辅助函数assign\_dispatch。
44. assign函数：接受一个参数std::initializer\_list；在实现上直接调用forward\_iterator版本的assign\_aux函数。
45. push\_back函数：接受一个参数value；如果缓冲区有一个以上的空间(cur != last - 1)，调用construct函数在最后一个位置创建元素，并调整finish迭代器cur，如果缓冲区只有一个空间，调用辅助函数push\_back\_aux。
46. push\_back函数：实现上和带参数的push\_back函数类似，只是不用指明value值。
47. push\_front函数：接受一个参数value；如果start的位置不是处于缓冲区的头部，则直接在start的前一个位置调用construct函数，并调整start的位置；如果start的位置处于缓冲区的头部，调用辅助函数push\_front\_aux。
48. push\_front函数：实现上和带参数的push\_front函数类似。
49. pop\_back函数：如果finish结点有元素，调整finish并调用destroy函数；如果finish结点没有元素，调用pop\_back\_aux函数。
50. pop\_front函数：如果当前头部元素不是处于结点的尾端，调用destroy函数并调整start；如果头部元素处于结点的尾端，调用pop\_front\_aux函数。
51. insert函数：接受两个参数position和value；如果position是start，直接调用push\_front函数并返回start，如果position是finish，调用push\_back函数并返回finish的前一个位置，否则调用辅助函数insert\_aux。(返回值是新数据的位置)
52. insert函数：接受一个参数position，调用接受两个参数的insert函数。
53. insert函数：接受三个参数position，n和value；调用辅助函数fill\_insert实现。
54. insert函数：模板函数，模板参数为InputIterator，接受三个参数position，first和last；利用Is\_integer判断模板参数是否为整数，调用辅助函数insert\_dispatch。
55. insert函数：接受三个参数position，first和last，其中first和last都是const value\_type\*类型；实现上和insert函数类似。
56. insert函数：接受三个参数position，first和last，其中first和last都是const\_iterator类型；实现上和insert函数类似。
57. erase函数：接受一个参数position；记录下position的下一个位置next，计算出position之前的元素个数index，如果前半部分元素较少，将前半部分元素后移一个位置，然后调用pop\_front函数；否则将后半部分元素前移一个位置，然后调用pop\_back函数；返回值类型为iterator，返回start+index。
58. erase函数：接受两个参数first和last；如果first和last覆盖了所有元素，直接调用clear函数并返回finish；记录下需要删除的元素个数n和前半部分元素个数elems\_before，如果前半部分元素少(注意这里的判断条件)，调用copy\_backward函数将start到first的元素移动到last，然后记录new\_start的值，调用destroy函数将start到new\_start的元素清除，调用destroy\_node将start到new\_start的结点清除；如果后半部分元素少，调用copy函数将last到finish的元素移动到first，记录new\_finish的值，然后分别调用destroy和destroy\_node函数；返回值类型为iterator，返回start+elems\_before。
59. clear函数(需要保留一个缓冲区)：利用for循环对first到last直接的缓冲区(不包括first和last)调用destroy和deallocate函数，如果start不等于last，调用destroy函数将start和finish中所有元素清除，然后调用deallocate函数清除finish缓冲区，如果start等于finish，调用destroy函数清除start到finish之间的所有元素，最后将finish赋值为start。
60. 辅助函数：
61. initialize\_map：接受一个参数num\_elements；先计算出需要的缓冲区个数num\_nodes(如果刚好整除会多分配一个)，然后map\_size等于initial\_map\_size(注意类型转换)和num\_nodes+2的最大值(+2是因为前后各预留一个，扩充时使用)；然后调用map\_allocator的allocate函数分配map\_size个空间，返回值给map；声明开始结点nstart(T\*\*)为map+(map\_size – num\_nodes) / 2(中间位置)，nfinish(最后一个结点的下一个位置)为nstart+num\_nodes；调用辅助函数create\_nodes创建结点，如果失败调用map\_allocator的deallocate函数，并把map设为空，map\_size设为0(这里使用try和catch实现)；调用set\_node函数将start和finish赋值；start的cur等于start的finish，finish的cur等于finish的first偏移num\_elements%deque\_buf\_size的位置。
62. create\_nodes：接受两个参数start和finish，都是T\*\*类型；利用for循环对每个结点调用data\_allocator的allocate函数，allocate函数的参数为deque\_buf\_size(一个缓冲区元素的个数)，如果失败调用辅助函数destroy\_nodes(这里使用try和catch实现)。
63. destroy\_nodes函数：接受两个参数start和finish，都是T\*\*类型；利用for循环对每个结点调用data\_allocator的deallocate函数，deallocate函数的第二个参数为deque\_buf\_size。
64. fill\_initialize函数：接受一个参数value；利用for循环对start到finish之前结点的所有结点调用uninitialized\_fill函数进行赋值，对finish结点的first到cur元素调用uninitialized\_fill函数进行赋值，如果失败调用destroy函数进行清除(这里使用try和catch实现，destroy的第二参数为临时的iterator，表示从start到失败的地点cur)。
65. initialize\_dispatch函数：模板函数，模板参数为Integer，接受三个参数n，value和true\_type；调用initialize\_map函数初始化map，然后调用fill\_initialize函数进行赋值。
66. initialize\_dispatch函数：模板函数，模板参数为InputIterator，接受三个参数first，last和false\_type；调用辅助函数range\_initialize(要分迭代器的类型)(为什么要分区，以及为什么要用ForwardIterator为名称，感觉应该是random\_access\_iterator在实现上才有区分)。
67. range\_initialize函数(input iterator版本)：模板函数，模板参数为InputIterator；调用initialize\_map函数初始化map，这里参数为0；然后利用for循环对first到last的元素进行push\_back，如果失败调用clear函数(这里使用try和catch实现)。
68. range\_initialize函数(forward iterator版本)：模板函数，模板参数为ForwardIterator；利用distance计算出元素个数n，调用initialize\_map函数初始化map，利用for循环对start到finish之前结点的所有结点进行uninitialize\_copy操作，每deque\_buf\_size个元素为一组，一组一组的进行，最后对finish结点的元素进行赋值，如果失败调用destroy函数，destroy函数的第二参数是一个临时的iterator代表当前结点cur(这里使用try和catch实现)。
69. fill\_assign：接受两个参数n和value；如果n大于当前deque的size，先调用fill函数将begin到end填充value，然后调用insert插入剩余的元素；如果n小于等于当前deque的size，先调用fill函数填充n的元素，然后调用erase函数删除剩余的元素。
70. assign\_dispatch函数：模板函数，模板参数为Integer，接受三个参数n，value和true\_type；实现上调用辅助函数fill\_assign。
71. assign\_dispatch函数：模板函数，模板参数为InputIterator，接受三个参数first，last和false\_type；实现上需要根据迭代器的类型做不同的操作，调用辅助函数assign\_aux。
72. assign\_aux函数(input iterator版本)：模板函数，模板参数为InputIterator，接受三个参数first，last和input\_iterator\_tag；实现上利用for循环进行插入，如果first等于last，调用erase清除剩余的元素，如果迭代器cur等于end，调用insert函数插入多出的元素。
73. assign\_aux函数：(forward iterator版本)：模板函数，模板参数为ForwardIterator，接受三个参数first，last和forward\_iterator\_tag；实现上先调用distance函数计算出first到last的距离len，如果len大于当前deque的size，先调用copy函数拷贝前部分元素到deque中，然后利用insert将剩余的元素插入；如果len小于等于当前deque的size，先调用copy函数将first到last的元素拷贝到deque中，然后调用erase函数清除剩余的元素。
74. push\_back\_aux函数：接受一个参数value；先调用辅助函数reserve\_map\_at\_back对map进行调整，然后调用data\_allocator的allocate函数对finish的下一个结点分配空间，然后利用construct函数进行赋值，调用set\_node函数设置finish的值，然后将finish的cur设置为finish的first，如果失败重新调整finish的node和cur的值，并destroy新元素的值，调用data\_allocator的deallocate函数释放新的结点(这里利用try和catch实现)。
75. push\_back\_aux函数：实现上和带一个参数的push\_back\_aux函数类似。
76. push\_front\_aux函数：接受一个参数value；先调用函数reserve\_map\_at\_front调整map，然后调用data\_allocator的allocate函数为start的前一个结点分配空间，调用set\_node设置start结点的值，调整start cur的值并调用construct赋值，如果失败重新调整start的node和cur的值，并调用deallocate函数将失败的node清除。
77. push\_front\_aux函数：实现上和接受一个参数的push\_front\_aux函数类似。
78. reserve\_map\_at\_back函数：接受一个参数nodes\_to\_add(size\_type 默认为1)；如果尾部没有空的结点(这里的判断注意细节，尾部永远要多一个空的node)，调用辅助函数reallocate\_map(nodes\_to\_add, false)。这里的第二个参数表示插入的位置是在头部(true)或尾部(false)。
79. reserve\_map\_at\_front函数：接受一个桉树nodes\_to\_add(size\_type 默认为1)；如果头部没有空的结点(注意类型转换)，调用辅助函数reallocate\_map(nodes\_to\_add, true)。这里的第二个参数表示插入的位置是在头部(true)或尾部(false)。
80. reallocate\_map函数：接受两个参数nodes\_to\_add和add\_at\_front；先计算出原来的结点数和新的结点数，如果当前map的大小大于2倍新结点个数大小(这个条件是确保new\_start+new\_num\_nodes不会越界)，计算出new\_start的位置(先放正中间，然后加上需要在前面增加的结点数)，如果new\_start是在原来start结点的前面，调用copy函数将原来的结点拷贝到新的位置，否则利用copy\_backward函数进行拷贝(注意拷贝时尾端结点是finish+1)；如果map的大小小于等于2倍新结点个数大小，计算出新map的大小map\_size+max(map\_size，nodes\_to\_add) + 2，然后利用map\_allocator为新的map分配空间，计算出new\_start的位置，调用copy函数将原来的结点拷贝到新的map中，然后调用deallocate销毁原来的map，设置map和map\_size的值；调用set\_nodes函数调整start和finish的值。
81. pop\_back\_aux函数：先调用deallocate函数清除finish结点，然后设置finish的node和cur，调用destroy函数清除元素。
82. pop\_front\_aux函数：调用destroy函数清除元素，调用deallocate函数清除start结点，设置start结点的node和cur。
83. insert\_aux函数：接受两个参数position和value；先计算出插入点之前的元素个数index，如果index小于当前size的一半(前半部分元素较少)，先调用push\_front在最前面插入一个与第一个元素同值的元素，标记front1，front2的位置，重新计算position的位置(start+index)，标记position下一个位置为pos1，调用copy函数进行拷贝(copy(front2, pos1, front1))；如果后半部分元素较少，现在尾端push\_back一个与最后元素同值的元素，然后标记back1和back2的位置，重新计算position的位置，调用copy\_backward函数拷贝(copy(backward(pos, back2, back1)))；最后在position位置上赋值value并返回position。
84. fill\_insert函数：接受三个参数position，n和value；如果position为start，先调用辅助函数reserve\_elements\_at\_front并将返回值赋给new\_start，然后调用uninitialize\_fill进行赋值并将new\_start赋给start，如果失败destroy\_nodes掉new\_start到start之间的结点(这里使用try和catch实现)；如果position为finish，先调用辅助函数reserve\_elements\_at\_back并将返回值赋给new\_finish，然后调用uninitialize\_fill进行赋值并将new\_finish赋给finish，如果失败destroy\_nodes掉finish+1到new\_finish+1之间的结点(+1的原因：想清楚finish的含义)；其他情况调用insert\_aux函数。
85. reverse\_elements\_at\_front函数：接受一个参数n；计算出当前start前面的空间vacancies，如果小于n，调用辅助函数new\_elements\_at\_front，返回值类型为iterator，返回start前n个位置。
86. reverse\_elements\_at\_back函数：接受一个参数n；计算出当前finish之后的空间vacancies，如果小于n，调用辅助函数new\_elements\_at\_back，返回值类型为iterator，返回finish后n个位置。
87. new\_elements\_at\_front函数：接受一个参数new\_elements；先计算出需要增加的结点个数new\_nodes，调用reserve\_map\_at\_front函数，利用for循环对所有的新增结点进行allocate分配空间，如果失败对新分配空间的结点进行deallocate(用try和catch实现)。
88. new\_elements\_at\_back函数：接受一个参数new\_elements；实现是和new\_elements\_at\_front类型，调用的函数换为了reserve\_map\_at\_back，并且在allocate和deallocate时的+-有所改变。
89. insert\_aux函数：接受三个参数position，n和value；首先算出当前位置之前的元素个数elems\_before，如果前半部分元素较少(elems\_before小于size的一半)，调用reserve\_elements\_at\_front配置新空间，并将返回值赋给new\_start，记录原来start的位置old\_start，重新得到position的值(有可能在reserve\_elements\_at\_front中变化)，如果elems\_before大于等于n，先后调用uninitialized\_copy和copy将start到position的元素拷贝到前面，然后调用fill函数填充新数据；如果elems\_before小于n，先后调用uninitialized\_copy和uninitialized\_fill将start到position的元素拷贝到前面并进行填充，然后调用fill函数填充剩下的元素，如果失败调用destroy\_nodes函数将new\_start到start的结点清除(这里使用try和catch实现)；如果后半部分的元素更少，调用reserve\_elements\_at\_back配置新空间，记录原来finish的位置，计算出后半部分的元素个数elems\_after，重新计算出position的位置，如果elems\_after大于n，先后调用uninitialized\_copy和copy\_backward将原来的元素向后移动，然后调用fill函数填充新元素，如果elems\_after小于等于n，先后调用uninitialized\_fill和uninitialized\_copy向后拷贝并填充部分元素，然后利用fill函数填充剩下的元素，如果失败调用destroy\_nodes函数将finish+1到new\_finish+1的结点清除(这里使用try和catch实现)。
90. insert\_dispatch函数：模板函数，模板参数为Integer，接受四个参数position，n，value和true\_type；实现上调用辅助函数fill\_insert。
91. insert\_dispatch函数：模板函数，模板参数为InputIterator，接受四个参数position，first，last和false\_type；实现上根据迭代器的类型调用insert函数。
92. insert函数(input iterator版本)：模板函数，模板参数为InputIterator，接受四个参数position，first，last和input\_iterator\_tag；实现上调用copy函数，其中copy函数的第三个参数为inserter迭代器(insert(\*this, pos))。
93. insert函数(forward iterator版本)：模板函数，模板参数为ForwardIterator，接受四个参数position，first，last和forward\_iterator\_tag；实现上和insert函数类似。
94. insert\_aux函数：模板函数，模板参数为ForwardIterator，接受四个参数position，first，last和n；实现上和接受三个参数的insert\_aux函数类似。
95. insert\_aux函数：first和last为const value\_type\*版本，实现上和insert\_aux类似。
96. insert\_aux函数：first和last为const\_iterator版本，实现上和insert\_aux类似。
97. 常用函数：
98. operator==：接受两个参数lhs和rhs；判断两个deque的size是否相同并且调用equal函数判断每个元素是否相同。
99. operator!=：接受两个参数lhs和rhs；调用operator==。
100. operator<：接受两个参数lhs和rhs；调用lexicographical\_compare函数进行判断。
101. operator>：接受两个参数lhs和rhs；调用operator<。
102. operator<=：接受两个参数lhs和rhs；调用operator<。
103. operator>=：接受两个参数lhs和rhs；调用operator>。
104. swap函数：接受两个参数lhs和rhs；通过lhs调用swap函数进行交换。

### stack

1. stack

模板类，有两个模板参数T和Sequence(默认为deque)；typedef(都是用底层容器表示)包括container\_type，value\_type，size\_type，reference，const\_reference；维护一个底层容器c。

1. 构造、移动和析构函数
2. 默认构造函数：采用default。
3. 构造函数：接受一个参数n，显式声明；利用底层容器进行构造。
4. 构造函数：接受两个参数n和value；利用底层容器进行构造。
5. 构造函数：接受一个参数std::initializer\_list的ilist；利用底层容器进行构造。
6. 构造函数：模板函数，模板参数为InputIterator，接受两个参数first和last；利用底层容器进行构造。
7. 构造函数：接受一个参数容器c；利用底层容器进行构造。
8. 拷贝构造函数：接受一个参数rhs；利用底层容器进行构造。
9. 重载operator=：接受一个参数rhs；调用底层容器的operator=，返回值类型为stack&，返回自身。
10. 重载operator=：接受一个参数std::initializer\_list的ilist；调用底层容器的operator=，返回值类型为stack&，返回自身。
11. 析构函数：采用default。
12. 访问元素相关操作
13. top函数：调用底层容器的back函数，返回值类型为reference。
14. top函数：const成员函数，调用底层容器的back函数，返回值类型为const\_reference。
15. 容量相关操作
16. empty函数：调用底层容器的empty函数。
17. size函数；调用底层容器的size函数。
18. 修改容器相关操作
19. push函数：接受一个参数value；调用底层容器的push\_back函数。
20. pop函数：调用底层容器的pop\_back函数。
21. clear函数：利用while循环一直调用pop函数。
22. swap函数：接受一个参数rhs；调用底层容器的swap函数。
23. operator==：友元函数，接受两个参数lhs和rhs；调用底层容器的operator==。
24. operator<：友元函数，接受两个参数lhs和rhs；调用底层容器的operator<。
25. 常用函数
26. operator==：接受两个参数lhs和rhs；直接比较lhs和rhs。
27. operator<：接受两个参数lhs和rhs；直接比较lhs和rhs。
28. operator!=：接受两个参数lhs和rhs；调用operator==。
29. operator>：接受两个参数lhs和rhs；调用operator<。
30. operator<=：接受两个参数lhs和rhs；调用operator<。
31. operator>=：接受两个参数lhs和rhs；调用operator>。
32. swap函数：接受两个参数lhs和rhs；调用lhs的swap函数。

### queue

1. queue

模板类，有两个模板参数T和Sequence(默认为deque)；typedef(都是用底层容器表示)包括container\_type，value\_type，size\_type，reference，const\_reference；维护一个底层容器c。

1. 构造、移动和析构函数
2. 默认构造函数：采用default。
3. 构造函数：接受一个参数n，显式声明；利用底层容器进行构造。
4. 构造函数：接受两个参数n和value；利用底层容器进行构造。
5. 构造函数：接受一个参数std::initializer\_list的ilist；利用底层容器进行构造。
6. 构造函数：模板函数，模板参数为InputIterator，接受两个参数first和last；利用底层容器进行构造。
7. 构造函数：接受一个参数容器c；利用底层容器进行构造。
8. 拷贝构造函数：接受一个参数rhs；利用底层容器进行构造。
9. 重载operator=：接受一个参数rhs；调用底层容器的operator=，返回值类型为stack&，返回自身。
10. 重载operator=：接受一个参数std::initializer\_list的ilist；调用底层容器的operator=，返回值类型为stack&，返回自身。
11. 析构函数：采用default。
12. 访问元素相关操作
13. front函数：调用底层容器的front函数，返回值类型为reference。
14. front函数：const成员函数，调用底层容器的front函数，返回值类型为const\_reference。
15. back函数：调用底层容器的back函数，返回值类型为reference。
16. back函数：const成员函数，调用底层容器的back函数，返回值类型为const\_reference。
17. 容量相关操作
18. empty函数：调用底层容器的empty函数。
19. size函数；调用底层容器的size函数。
20. 修改容器相关操作
21. push函数：接受一个参数value；调用底层容器的push\_back函数。
22. pop函数：调用底层容器的pop\_front函数。
23. clear函数：利用while循环一直调用pop函数。
24. swap函数：接受一个参数rhs；调用底层容器的swap函数。
25. operator==：友元函数，接受两个参数lhs和rhs；调用底层容器的operator==。
26. operator<：友元函数，接受两个参数lhs和rhs；调用底层容器的operator<。
27. 常用函数
28. operator==：接受两个参数lhs和rhs；直接比较lhs和rhs。
29. operator<：接受两个参数lhs和rhs；直接比较lhs和rhs。
30. operator!=：接受两个参数lhs和rhs；调用operator==。
31. operator>：接受两个参数lhs和rhs；调用operator<。
32. operator<=：接受两个参数lhs和rhs；调用operator<。
33. operator>=：接受两个参数lhs和rhs；调用operator>。
34. swap函数：接受两个参数lhs和rhs；调用lhs的swap函数。

### Priority\_queue

1. priority\_queue

模板类，模板参数为T，Sequence(默认为vector)，Compare(默认为less<typename Sequence::value\_type)；typedef(都是用底层容器表示)包括container\_type，value\_type，size\_type，reference，const\_reference；维护一个底层容器c和comp。

1. 构造、移动和析构函数
2. 默认构造函数：采用default。
3. 构造函数：接受一个参数c(Compare)；将c赋值给comp。
4. 构造函数：接受一个参数n，显式声明；利用底层容器进行构造，调用make\_heap函数。
5. 构造函数：接受两个参数n和value；利用底层容器进行构造，调用make\_heap函数。
6. 构造函数：接受一个参数std::initializer\_list的ilist；利用底层容器进行构造。
7. 构造函数：模板函数，模板参数为InputIterator，接受两个参数first和last；利用底层容器进行构造，调用make\_heap函数。
8. 构造函数：接受一个参数容器c；利用底层容器进行构造，调用make\_heap函数。
9. 拷贝构造函数：接受一个参数rhs；利用底层容器进行构造，调用make\_heap函数。
10. 重载operator=：接受一个参数rhs；调用底层容器的operator=，调用make\_heap函数，返回值类型为priority\_queue&，返回自身。
11. 重载operator=：接受一个参数std::initializer\_list的ilist；调用底层容器的operator=，调用make\_heap函数，返回值类型为priority\_queue&，返回自身。
12. 析构函数：采用default。
13. 访问元素相关操作
14. top函数：const成员函数，调用底层容器的front函数，返回值类型为const\_reference。
15. 容量相关操作
16. empty函数：调用底层容器的empty函数。
17. size函数；调用底层容器的size函数。
18. 修改容器相关操作
19. push函数：接受一个参数value；调用底层容器的push\_back函数，调用push\_heap函数。
20. pop函数：调用pop\_heap函数，调用底层容器的pop\_back函数。
21. clear函数：利用while循环一直调用pop函数。
22. swap函数：接受一个参数rhs；调用swap函数交换c和comp。
23. operator==：友元函数，接受两个参数lhs和rhs；调用底层容器的operator==。
24. operator!=：友元函数，接受两个参数lhs和rhs；调用底层容器的operator!=。
25. 常用函数
26. operator==：接受两个参数lhs和rhs；直接比较lhs和rhs。
27. operator!=：接受两个参数lhs和rhs；调用operator==。
28. swap函数：接受两个参数lhs和rhs；调用lhs的swap函数。

## 关联式容器

由于在rb\_tree中需要用到pair，因此首先在文件mystl\_pair中实现pair。

1. pair

是一个模板类，有两个模板参数T1和T2；typedef将T1定义为first\_type，T2定义为second\_type。拥有两个成员first和second。

1. 构造和复制函数
2. 默认构造函数：将first赋值为T1的默认值，second赋值为T2的默认值。
3. 构造函数：接受两个参数a和b；将a赋值给first，b赋值给second。
4. 复制构造函数：是一个模板函数，有两个模板参数U1和U2；接受一个参数p(pair<U1，U2>)；将p.first赋值给first，p.second赋值给second。
5. 其他函数(都是内联函数)
6. operator==：模板函数，有两个模板参数T1和T2；接受两个参数lhs和rhs；返回lhs.first等于rhs.first并且lhs.second等于rhs.second。
7. operator<：模板函数，有两个模板参数T1和T2；接受两个参数lhs和rhs；返回lhs.first小于rhs.first或者lhs.first等于rhs.first(注意这里的实现)并且lhs.second小于rhs.second。
8. operator！=：模板函数，有两个模板参数T1和T2；接受两个参数lhs和rhs；调用operator==。
9. operator>：模板函数，有两个模板参数T1和T2；接受两个参数lhs和rhs；调用operator<。
10. operator<=：模板函数，有两个模板参数T1和T2；接受两个参数lhs和rhs；调用operator<。
11. operator>=：模板函数，有两个模板参数T1和T2；接受两个参数lhs和rhs；调用operator<。
12. make\_pair函数：模板函数，有两个模板参数T1和T2；接受两个参数x和y；调用pair的构造函数并返回；返回值类型为pair<T1，T2>。

### RB\_tree

1. 红黑树结点颜色类型定义

利用typedef将红黑树的颜色类型定义为bool，并重新命令为rb\_tree\_color\_type，利用静态常量表达式将红色结点定义为false，黑色结点定义为true。

1. 红黑树结点设计
2. 结点分为rb\_tree\_node\_base和rb\_tree\_node两部分，都是模板类。
3. rb\_tree\_node\_base：typedef包括color\_type，base\_ptr和node\_ptr；包括四个属性：父结点，左孩子结点，右孩子结点和结点颜色；成员函数包括get\_base\_ptr，get\_node\_ptr(使用reinterpret\_cast进行转换)和get\_node\_ref(使用reinterpret\_cast进行转换)。
4. rb\_tree\_node：typedef包括base\_ptr和node\_ptr；包括一个属性结点值value；成员函数包括get\_base\_ptr(用static\_cast进行转换)和get\_node\_ptr。
5. 红黑树迭代器设计
6. 迭代器分为rb\_tree\_iterator\_base，rb\_tree\_iterator和rb\_tree\_const\_iterator三部分，都是模板类。
7. rb\_tree\_iterator\_base：继承通用迭代器模板(迭代器类型为bidirectional\_iterator\_tag，类型为T)；typedef包括base\_ptr；包含一个base\_ptr类型的结点node；成员函数包括increment和decrement；increment函数分为两种情况，如果当前结点的右孩子结点不为空，则下一结点就是右孩子结点的最左结点(这里可以使用一个通用算法实现)；如果当前结点的右孩子结点为空，则先寻找当前结点的最左上结点，最左上结点的父结点就是下一结点(这里需要考虑特殊情况：只有根节点；只有当node->rchild!=y时，y结点才是下一结点)；decrement函数分为三种情况：当前结点为头结点，此时前一结点就是最大结点node->right，如果当期结点的左孩子结点不为空，则前一结点就是左子树的最右结点(这里可以使用一个通用算法实现)，如果当前结点既不是头结点，也没有左孩子结点，则前一结点为最右上结点的父结点；重载==和!=两种操作符，直接对node进行比较。
8. rb\_tree\_iterator：继承rb\_tree\_iterator\_base；typedef包括value\_type，pointer，reference，base\_ptr，node\_ptr，iterator，const\_iterator和self；利用using语句使用rb\_tree\_iterator\_base中的node；构造函数包括五种：默认构造函数，参数为base\_ptr的构造函数，参数为node\_ptr的构造函数，参数为const iterator的构造函数和参数为const\_iterator的构造函数；重载操作符包括：operator\*，利用node获得node\_ptr的指针再获取值，返回值为reference；operator->，调用operator->，返回值为pointer；operator++，调用increment函数，返回值为self&；operator--，调用decrement函数，返回值为self&；operator++(int)和operato--(int)。
9. rb\_tree\_const\_iterator：类似rb\_tree\_iterator。
10. 算法
11. rb\_tree\_min：模板函数，模板参数为Node\_ptr；接受一个参数x，如果当前结点的左孩子结点不为空，找到最左下结点并返回，否则返回x，返回值类型为Node\_ptr。
12. rb\_tree\_max：模板函数，模板参数为Node\_ptr；接受一个参数x，如果当前结点的右孩子结点不为空，找到最右下结点并返回，否则返回x，返回值类型为Node\_ptr。
13. rb\_tree\_is\_lchild：模板函数，模板参数为Node\_ptr；接受一个参数node，判断node是否为左孩子结点，返回值为bool。
14. rb\_tree\_is\_rchild：模板函数，模板参数为Node\_ptr；接受一个参数node，判断node是否为右孩子结点，返回值为bool。
15. rb\_tree\_is\_red：模板函数，模板参数为Node\_ptr；接受一个参数node，判断当前结点的颜色是否为红色，返回值为bool。
16. rb\_tree\_is\_black：模板函数，模板参数为Node\_ptr；接受一个参数node，判断当前结点的颜色是否为黑色，返回值为bool。
17. rb\_tree\_set\_red：模板函数，模板参数为Node\_ptr；接受一个参数node&，将当前结点的颜色设为红色。
18. rb\_tree\_set\_black：模板函数，模板参数为Node\_ptr；接受一个参数node&，将当前结点的颜色设为黑色。
19. rb\_tree\_next：模板函数，模板参数为Node\_ptr；接受一个参数node，如果当前结点右孩子结点不为空，则返回右子树的最左结点；如果当前结点右孩子结点为空，返回最左上结点的父结点，返回值类型为Node\_ptr。
20. rb\_tree\_rotate\_left：模板函数，模板参数为Node\_ptr；接受两个参数x和root；首先将x的右孩子结点设为y，把y的左子树设置为x的右子树，并且如果y的左子树不为空，修改y的左子树的父结点，修改y的父结点，如果x为根结点，将y设置成新的根结点，如果x为左孩子结点，将x的父结点的左子树设为y，如果x为右孩子结点，将x的父结点的右子树设为y，最后调整x与y的关系。
21. rb\_tree\_rotate\_right：和rb\_tree\_rotate\_left类似。
22. rb\_tree\_insert\_rebalance：模板函数，模板参数为Node\_ptr；接受两个参数x和root(引用)；分为五种情况讨论：
23. 新增结点位于根结点；令新增结点为黑色。
24. 新增结点的父结点为黑色；没有破坏平衡，直接返回。
25. 新增结点的父结点和叔叔结点都为红色(爷结点一定为黑色)；令父结点和叔叔结点为黑色，爷结点为红色，并令爷结点为当前结点。
26. 新增结点的父结点为红色，叔叔结点为黑色，父结点为左(右)孩子结点，当前结点是父结点的右(左)结点；令当前结点为父结点，再以当前结点为支点进行左(右)旋。
27. 新增结点的父结点为红色，叔叔结点为黑色，父结点为左(右)孩子结点，当前结点是父结点的左(右)结点；令父结点为黑色，叶结点为红色，再以叶结点为支点进行右(左选)。

这五种情况的逻辑关系：主要考虑是否会破坏性质4(不能出现连续两个红色结点)和性质5(对于任意结点而言，其到叶结点树尾端NIL指针的每条路径都包含相同数目的黑结点)；情况a最简单；情况b没有破坏两个性质；情况c破坏了性质4，因此需要将父结点染为黑色，但此时会破坏性质5，因此也需要将爷结点染为红色，但爷结点右子树包含的黑结点个数会减少1个(破坏性质5)，因此将叔叔结点染为红色即可解决；情况d主要是为了将红黑树调整为情况e，此时的旋转操作不会破坏性质5；情况e不能采取情况c做法的原因就是因为叔叔结点的颜色为黑色，如果将父结点染为黑色，叶结点染为红色，此时叶结点的右子树包含的黑结点个数会少1，为了解决这种情况，只能将父结点作为支点进行相应的旋转操作，此时原叶结点左子树的黑色结点个数不变，右子树的黑色结点个数加1。(参考资料：<https://blog.csdn.net/v_JULY_v/article/details/6105630>)

1. rb\_tree\_erase\_rebalance(太难了)：模板函数，模板参数为Node\_ptr；接受四个参数z，root(Node\_ptr&)，leftmost(Node\_ptr&)和rightmost(Node\_ptr&)；首先考虑删除结点z(思想为用z结点的孩子结点或后继结点代替z结点)：声明y为新的待删除结点，x为y结点的其中一个孩子结点，xp为x的父结点；如果y不等于z，说明此时z结点有两个非空子结点，y指向z的右子树的最左下结点，x为y的右孩子结点，首先用y结点取代z结点(重建y结点和z结点的左孩子结点之间的指针)，如果y不是z的右孩子结点，用x结点代替y结点(重建x结点和y结点的父结点之间的指针(这里注意x结点可能为空)，将xp设为y的父结点)，重建y结点和z结点的右孩子结点之间的指针，如果y结点是z结点的右孩子结点，直接将xp设为y；如果z结点为根结点，将y结点设为根结点，重建y结点和z结点的父结点之间的指针(注意区分左右孩子)，交换y和z的颜色，将y结点设为z结点；如果y等于z，此时z结点最多只有一个孩子结点(x结点，有可能为空)，用x结点取代y结点(将xp设为y的父结点，重建x结点和y的父结点之间的指针(注意左右孩子以及x为空的情况))，如果z是根结点，将x设为根结点，此时的z可能是最大或最小结点，如果z是最小结点(如果x为空，最小结点设为xp，否则设为x子树的最小结点)，如果z是最大结点(如果x为空，最大结点设为xp，否则设为x子树的最大结点)；接下来讨论删除结点之后对红黑树的重新调整，此时y指向要删除的结点，x为替代的结点，分为五种情况：
2. 删除结点的颜色为红色，红黑树的性质没有被破坏，不用进行操作。
3. 红黑树

模板类，有五个模板参数Key，Value，Keyofvalue，Compare和Alloc。

1. typedef：包括base\_ptr，base\_type，node\_ptr，node\_type，key\_type，value\_type，key\_compare，allocator\_type，data\_allocator，base\_allocator，node\_allocator，pointer，const\_pointer，reference，const\_reference，size\_type，difference\_type，iterator，const\_iterator，reverse\_iterator，const\_reverse\_iterator和link\_type。
2. 成员变量：header(base\_ptr)，node\_count(size\_type)，key\_cmp(key\_compare)。
3. 三个私有函数(返回值都为base\_ptr&)：root函数用于取得根结点；leftmost函数用于取得最小结点；rightmost函数用于取得最大结点。
4. 构造、复制和析构函数：
5. 默认构造函数：将头结点设为空，结点数量设为0，调用辅助函数get\_allocator为头结点分配空间，然后调用辅助函数empty\_initialize。
6. 构造函数：接受一个参数comp；将头结点设为空，结点数量设为0，comp设为参数值，调用辅助函数get\_allocator为头结点分配空间，然后调用辅助函数empty\_initailize。
7. 复制构造函数：接受一个参数rhs；将头结点设为空，结点数量设为0，comp设为rhs的comp，调用辅助函数get\_node为头结点分配空间，如果rhs的root为空，调用empty\_initialize函数，否则将头结点设为红色，将root设为调用辅助函数copy\_from的返回值，然后将leftmost设为rb\_tree\_min函数的返回值，rightmost设为rb\_tree\_max的返回值，最后将结点个数复制。
8. operator=：接受一个参数rhs；如果rhs就是当前红黑树，则直接返回this；首先调用clear函数，将结点个数设为0，并把key\_comp设置为rhs的比较函数，如果rhs的根结点为空，将root设为空，并把左右孩子指针指向头结点；如果rhs的根结点不为空，将copy\_from函数的返回值赋给root ，然后将leftmost设为rb\_tree\_min函数的返回值，rightmost设为rb\_tree\_max的返回值，最后将结点个数复制。
9. 析构函数：调用clear函数。
10. 迭代器相关函数：
11. begin函数：调用leftmost函数，返回值类型为iterator。
12. begin函数：const成员函数；调用leftmost函数，返回值类型为const\_iterator。
13. end函数：返回header，返回值类型为iterator。
14. end函数：const成员函数，返回header，返回值类型为const\_iterator。
15. rbegin函数：返回end函数的reverse\_iterator；返回值类型为reverse\_iterator。
16. rbegin函数：const成员函数，返回end函数的reverse\_iterator；返回值类型为const\_reverse\_iterator。
17. rend函数：返回begin函数的reverse\_iterator；返回值类型为reverse\_iterator。
18. rend函数：const成员函数，返回begin函数的reverse\_iterator；返回值类型为const\_reverse\_iterator。
19. cbegin函数：const成员函数，调用begin函数，返回值类型为const\_iterator。
20. cend函数：const成员函数，调用end函数，返回值类型为const\_iterator。
21. crbegin函数：const成员函数，调用rbegin函数，返回值类型为const\_reverse\_iterator。
22. crend函数：const成员函数，调用rend函数，返回值类型为const\_reverse\_iterator。
23. 容量相关函数
24. empty函数：判断node\_count是否为0。
25. size函数：返回node\_count的值，返回值类型为size\_type。
26. max\_size函数：返回最大的个数，返回值类型为size\_type。
27. 结点相关函数：
28. create\_node函数：接受一个参数x(value\_type)；首先调用get\_node函数得到一个结点，在利用construct函数对结点进行赋值，如果失败调用put\_node函数(这里使用try和catch来实现)，返回值类型为node\_ptr，返回创建的结点。
29. clone\_node函数：接受一个参数x(node\_ptr)；首先调用create\_node函数创建一个新的结点，然后将新结点的color赋值为x的color，左右孩子指针设为空，返回值类型为node\_ptr，返回新创建的结点。
30. destroy\_node函数：接受一个参数p(node\_ptr)；首先调用destroy函数将p的值销毁，然后调用put\_node函数。
31. 插入删除函数
32. insert\_equal函数：接受一个参数value；将头结点声明为y结点(y的意义是x的父结点)，根结点声明为x结点，利用一个while循环，当x不为空时，将y设为x，x的值根据x的key和value的key决定(这里调用key\_comp函数，value的key通过KeyOfValue声明出一个临时对象然后调用函数获得，x的key类似，只是传入的参数为x的value值)；根据比较结果将x移动至x的左孩子或右孩子；最后调用insert函数，返回值类型为iterator，返回insert函数的返回值。
33. insert\_equal函数：接受两个参数position(iterator)和value；首先判断插入元素的位置，如果插入位置是最左端，如果size值大于0并且value的key小于等于插入结点的key值，调用insert函数进行插入，否则调用insert\_equal函数进行插入并返回；如果插入位置是最右侧(iterator的node等于头结点)，如果插入结点的key小于等于value的key值，调用insert函数进行插入，否则调用insert\_equal函数进行插入并返回；如果插入位置在其他位置，判断value的key是否位于插入结点和插入结点前一结点(before结点)的key值之间(闭区间)，如果为真且before结点的右孩子指针为空，调用insert函数并返回，如果before结点的右孩子指针不为空，则调用insert函数并返回(传入的参数不同)；如果key值不处于中间，调用insert\_equal函数并返回。
34. insert\_equal函数：是一个模板函数，模板参数为InputIterator，接受两个参数first和last；利用for循环对first到last之间的iterator调用insert\_equal进行插入。
35. insert\_unique函数：接受一个参数value；将头结点声明为y(y的意义为x的父结点)，根结点声明为x，声明一个bool变量comp为true，利用while循环，当x不为空时，y设为x，comp的值为value的key和x的key比较大小的值，如果为左就将x设为x的左孩子，否则设为x的右孩子；利用y结点声明一个迭代器j，如果comp为真，如果j为最左结点(begin函数)，调用insert函数并搭配true返回一个pair值；如果j不是最左结点，将j--变为上一结点，如果value的key大于j的key值，调用insert函数并搭配true返回一个pair值，否则将j和false搭配返回一个pair值。
36. insert\_unique函数：接受两个参数position(iterator)和value；实现上和带两个参数的insert\_equal函数相似，只是比较都换成了开区间，调用的insert\_equal函数改为了调用insert\_unique函数。
37. insert\_unique函数：是一个模板函数，模板参数为InputIterator，接受两个参数first和last；利用for循环对first到last之间的iterator调用insert\_unique进行插入。
38. insert函数：接受三个参数x(base\_ptr)，y(base\_ptr)和value；重新声明x和y结点类型为node\_ptr，再调用create\_node函数创造一个z结点，先考虑在左侧插入的情况，如果y为头结点或者x不为空或者value的key小于y的key值，将y的左孩子指针指向z，如果y为头结点，将根结点和rightmost都指向z，否则如果y是leftmost，将z设为leftmost；如果在右侧进行插入，将y的右孩子指针指向z，如果y为rightmost，将rightmost设为z；将z结点的父结点设为z，左右孩子指针设为空，调用rb\_tree\_insert\_rebalance调整红黑树的平衡，将结点个数+1，返回值类型为iterator，返回通过z结点创建的临时iterator。
39. erase函数：接受一个参数position；调用rb\_tree\_erase\_rebalance函数并将返回值设为y结点，调用destroy\_node函数将y结点清除，将结点个数减1。
40. erase函数：接受一个参数x(key\_type)；调用equal\_range函数获得区间边界，利用distance函数算出区间包含的结点个数，调用erase函数进行清除；返回值类型为size\_type，返回n。
41. erase函数：接受两个参数first(iterator)和last(iterator)；如果first和last包含的范围是整个树，直接调用clear函数，否则利用while循环调用erase函数进行清除。
42. erase函数：接受两个参数first(key\_type\*)和last(key\_type\*)；利用while循环调用erase函数进行清除。
43. erase函数(不考虑树的平衡)：接受一个参数x(node\_ptr)；利用while循环，当x不为0时，先递归删除x的右子树，然后将x的左子树设为结点y，调用destroy\_node函数删除结点x，将x设为y。
44. clear函数：如果结点个数不为0，传入根节点并调用erase函数递归删除整棵树，将leftmost和rightmost都指向头结点，根结点设为空，结点个数设为0。
45. 红黑树相关函数
46. find函数：接受一个参数k(key\_type)；声明一个y结点(大于等于k的第一个结点)和x结点(当前搜索结点)，利用一个while循环，当x不为空时，如果x的key值大于等于k，则y等于x，x等于x的左孩子；否则x等于x的右孩子；利用y结点声明一个迭代器j，如果j等于end或者j的key值大于k，返回end搜索失败，否则返回迭代器j。
47. find函数：接受一个参数k(key\_type)：和上述find函数类似，只是返回值类型为const\_iterator，只需把代码中的iterator改为const\_iterator即可。
48. count函数：接受一个参数k(key\_type)；调用equal\_range函数并将返回值设为p，利用distance函数计算区间的结点个数n；返回值类型为size\_type，返回n。
49. lower\_bound函数：接受一个参数k(key\_type)；声明一个结点y(大于等于k的第一个结点)和结点x(当前结点)，利用while循环，当x不为空时，如果x的key值大于等于k，将y设为x，x设为x的左子树，否则将x设为x的右子树，最后利用结点y声明一个迭代器j；返回值类型为iterator，返回j。
50. lower\_bound函数：接受一个参数k(key\_type)；和上述lower\_bound函数类似，只是返回类型为const\_iterator，只需把代码中的iterator改为const\_iterator即可。
51. upper\_bound函数：接受一个参数k(key\_type)；声明一个结点y(大于k的第一个结点)和结点x(当前结点)，利用while循环，当x不为空时，如果x的key值大于k，将y设为x，x设为x的左孩子，否则将x设为x的右孩子，利用结点y声明一个迭代器j；返回值类型为iterator，返回j。
52. upper\_bound函数：接受一个参数k(key\_type)；和上述upper\_bound函数类似，只是返回类型为const\_iterator，只需把代码中的iterator改为const\_iterator即可。
53. equal\_range函数：接受一个参数k(key\_type)；分别调用lower\_bound函数和upper\_bound函数获得两个迭代器，并利用这两个迭代器构造一个pair并返回。
54. equal\_range函数：接受一个参数k(key\_type)；和上述equal\_range函数类似，返回值变为了const\_iterator的pair，只需把代码中的iterator改为const\_iterator即可。
55. 辅助函数
56. get\_node函数：利用base\_allocator的allocate函数分配空间，返回值类型为base\_ptr。
57. put\_node函数：接受一个参数p(node\_ptr类型)；调用base\_allocator的deallocate函数进行结点的销毁。
58. empty\_initialize函数：将头结点的color设为红色，父结点设为空，左右孩子结点设为自己。
59. copy\_from函数：接受两个参数x和p(x的父结点)；首先调用clone\_node函数得到x结点的复制结点top，并将top的parent设为p，如果x的右子树不为空，递归调用copy\_from进行复制，将p设为top，x设为x的左孩子，利用while循环，当x不为空时，利用clone\_node函数得到x的复制结点y，将p的左孩子设为y，y的parent设为p，如果x的右孩子不为空，递归调用copy\_from函数进行复制，将p设为y，x设为x左孩子；如果失败调用erase\_since函数进行清除(这里使用try和catch进行)；返回值类型为base\_ptr，返回top。
60. erase\_since函数：接受一个参数x；利用while循环，当x不为空时，递归调用erase\_since删除x的右子树，设x的左孩子为y，利用destroy\_node(这里传入的是node不是base\_node)函数删除结点，将x设为y。
61. black\_count函数(从根结点到参数结点路径上黑色结点个数)：内联函数，接受两个参数node和root；如果node为空，返回0；声明一个变量bc，如果当前结点为黑色则bc为1否则为0，如果node为根结点，返回bc，否则递归调用black\_count函数，并将返回值和bc相加一起返回，返回值类型为int。
62. rb\_tree\_verity函数：如果结点个数为0或begin等于end，返回结点个数为0并且begin等于end并且头结点的左右结点都等于头结点的逻辑值；调用black\_count函数计算出leftmost到root之间黑结点的个数len，利用for循环对begin到end之间的结点进行遍历，声明三个迭代器x(当前结点)，L(当前结点的左孩子)，R(当前结点的右孩子)，如果当前结点为红色，如果左右孩子存在并且其中一个为红色，返回false；如果左右孩子存在并且和x的大小关系不对，返回false；如果左右孩子都不存在但是从x到root的黑结点个数不等于len，返回false；如果leftmost(rightmost)不等于root的min(max)，返回false；否则返回true。
63. swap函数：接受一个参数rhs；如果rhs不等于当前红黑树，利用mystl的swap交换三个成员变量。
64. 其他函数
65. operator==：接受两个参数x(rb\_tree)和y(rb\_tree)；判断两个红黑树的size是否相等并调用mytl的equal函数判断每个结点是否相等。
66. operator<：接受两个参数x和y；调用mystl的lexicographical\_compare函数比较两个红黑树。
67. operator>：接受两个参数x和y；调用operator<。
68. operator!=：接受两个参数x和y；调用operator==。
69. operator<=：接受两个参数x和y；调用operator<。
70. operator>=：接受两个参数x和y；调用operator<。
71. swap函数：接受两个参数x和y；通过x调用成员函数swap。

### set

1. set

模板类set，键值不允许重复；有三个模板参数Key，Compare(默认为less)和Alloc；typedef包括key\_type，value\_type，key\_compare和value\_type；模板成员包括一个底层为红黑树的树tree。

1. typedef

包括pointer，const\_pointer，reference，const\_reference，iterator，const\_iterator，reverse\_iterator，const\_reverse\_iterator，size\_type，difference\_type，allocator\_types；全部都是通过底层红黑树定义的型别。

1. 构造和复制函数
2. 默认构造函数：将比较函数Compare传给红黑树进行构造。
3. 构造函数(声明为explicit)：接受一个参数comp；将comp传给红黑树进行构造。
4. 模板构造函数：有一个模板参数InputIterator；接受两个参数first和last；调用红黑树的默认构造函数，然后调用红黑树的insert\_unique函数进行构造。
5. 模板构造函数：有一个模板参数InputIterator；接受三个参数first，last和comp；调用红黑树的构造函数，然后调用红黑树的insert\_unique函数进行构造。
6. 构造函数：接受一个参数ilist(initializer\_list)；调用红黑树的默认构造函数，然后调用红黑树的insert\_unique函数进行构造。
7. 复制构造函数：接受一个参数rhs；调用红黑树的复制构造函数。
8. operator=：接受一个参数rhs；将rhs的红黑树赋给当前容器红黑树，返回\*this；返回值类型为set&。
9. operator=：接受一个参数ilist(initializer\_list)；先调用clear函数将当前红黑树清除，然后调用insert\_unique函数进行构造；返回值类型为set&。
10. 接口函数
11. key\_comp函数：const成员函数；调用红黑树的key\_comp并返回，返回值类型为key\_comp。
12. value\_comp函数：const成员函数；调用红黑树的key\_comp并返回，返回值类型为value\_comp。
13. get\_allocator函数：const成员函数；调用红黑树的get\_allocator函数并返回；返回值类型为allocator\_type。
14. 迭代器相关函数
15. begin函数：调用红黑树的begin函数并返回；返回值类型为iterator。
16. begin函数：const成员函数；调用红黑树的begin函数并返回；返回值类型为const\_iterator。
17. end函数：调用红黑树的end函数并返回；返回值类型为iterator。
18. end函数：const成员函数；调用红黑树的end函数并返回；返回值类型为const\_iterator。
19. rbegin函数：利用end函数构造reverse\_iterator；返回值类型为reverse\_iterator。
20. rbegin函数：const成员函数；利用end函数构造reverse\_iterator；返回值类型为const\_reverse\_iterator。
21. rend函数：利用begin函数构造reverse\_iterator；返回值类型为reverse\_iterator。
22. rend函数：const成员函数；利用begin函数构造reverse\_iterator；返回值类型为const\_reverse\_iterator。
23. cbegin函数：const成员函数；调用begin函数并返回；返回值类型为const\_iterator。
24. cend函数：const成员函数；调用end函数并返回；返回值类型为const\_iterator。
25. crbegin函数：const成员函数；调用rbegin函数并返回；返回值类型为const\_reverse\_iterator。
26. crend函数：const成员函数；调用rend函数并返回；返回值类型为const\_reverse\_iterator。
27. 容量相关函数
28. empty函数：const成员函数；调用红黑树的empty函数并返回；返回值类型为bool。
29. size函数：const成员函数；调用红黑树的size函数并返回；返回值类型为size\_type。
30. max\_size函数：const成员函数；调用红黑树的max\_size函数并返回；返回值类型为size\_type。
31. swap函数：接受一个参数rhs；调用红黑树的swap函数。
32. 插入删除相关函数
33. insert函数：接受一个参数x(value\_type)；调用红黑树的insert\_unique函数进行插入并返回；返回值类型为pair<iterator，bool>。
34. insert函数：接受两个参数position和x；调用红黑树的insert\_unique函数进行插入并返回；返回值类型为pair<iterator，bool>。
35. insert函数：模板函数，模板参数为InputIterator；接受两个参数first和last；调用红黑树的insert\_unique函数进行插入。
36. erase函数：接受一个参数position；调用红黑树的erase函数进行消除。
37. erase函数：接受一个参数key(key\_type)；调用红黑树的erase函数并返回；返回值类型为size\_type。
38. erase函数：接受两个参数first和last；调用红黑树的erase函数进行清除。
39. clear函数：调用红黑树的clear函数进行清除。
40. set相关操作函数
41. find函数：const成员函数；接受一个参数key(key\_type)；调用红黑树的find函数并返回；返回值类型为iterator。
42. count函数：const成员函数；接受一个参数key(key\_type)；调用红黑树的find函数并判断返回值是否是end，如果是返回0，否则返回1；返回值类型为size\_type。
43. lower\_bound函数：const成员函数；接受一个参数key(key\_type)；调用红黑树的lower\_bound函数并返回；返回值类型为iterator。
44. upper\_bound函数：const成员函数；接受一个参数key(key\_type)；调用红黑树的upper\_bound函数并返回；返回值类型为iterator。
45. equal\_range函数：const成员函数；接受一个参数key(key\_type)；调用红黑树的equal\_range函数并返回；返回值类型为pair<iterator，iterator>。
46. 友元函数
47. operator==：接受两个参数lhs和rhs；比较lhs和rhs的红黑树是否相等。
48. operator<：接受两个参数lhs和rhs；比较lhs的红黑树是否小于rhs的红黑树。
49. 其他函数
50. operator==：接受两个参数rhs和lhs；比较rhs和lhs是否相等。
51. operator<：接受两个参数rhs和lhs；比较lhs是否小于rhs。
52. operator!=：接受两个参数rhs和lhs；调用operator==。
53. operator>：接受两个参数rhs和lhs；调用operator<。
54. operator<=：接受两个参数rhs和lhs；调用operator<。
55. operator>=：接受两个参数rhs和lhs；调用operator<。
56. swap函数：接受两个参数rhs和lhs；利用lhs调用swap函数。

### multiset

1. multiset

模板类multiset，键值允许重复；有三个模板参数Key，Compare(默认为less)和Alloc；typedef包括key\_type，value\_type，key\_compare和value\_type；模板成员包括一个底层为红黑树的树tree。

1. typedef

包括pointer，const\_pointer，reference，const\_reference，iterator，const\_iterator，reverse\_iterator，const\_reverse\_iterator，size\_type，difference\_type，allocator\_types；全部都是通过底层红黑树定义的型别。

1. 构造和复制函数

1) 默认构造函数：将比较函数Compare传给红黑树进行构造。

2) 构造函数(声明为explicit)：接受一个参数comp；将comp传给红黑树进行构造。

3) 模板构造函数：有一个模板参数InputIterator；接受两个参数first和last；调用红黑树的默认构造函数，然后调用红黑树的insert\_equal函数进行构造。

4) 模板构造函数：有一个模板参数InputIterator；接受三个参数first，last和comp；调用红黑树的构造函数，然后调用红黑树的insert\_equal函数进行构造。

5) 构造函数：接受一个参数ilist(initializer\_list)；调用红黑树的默认构造函数，然后调用红黑树的insert\_equal函数进行构造。

6) 复制构造函数：接受一个参数rhs；调用红黑树的复制构造函数。

7) operator=：接受一个参数rhs；将rhs的红黑树赋给当前容器红黑树，返回\*this；返回值类型为multiset&。

8) operator=：接受一个参数ilist(initializer\_list)；先调用clear函数将当前红黑树清除，然后调用insert\_equal函数进行构造；返回值类型为multiset&。

4. 接口函数

1) key\_comp函数：const成员函数；调用红黑树的key\_comp并返回，返回值类型为key\_comp。

2) value\_comp函数：const成员函数；调用红黑树的key\_comp并返回，返回值类型为value\_comp。

3) get\_allocator函数：const成员函数；调用红黑树的get\_allocator函数并返回；返回值类型为allocator\_type。

5. 迭代器相关函数

1) begin函数：调用红黑树的begin函数并返回；返回值类型为iterator。

2) begin函数：const成员函数；调用红黑树的begin函数并返回；返回值类型为const\_iterator。

3) end函数：调用红黑树的end函数并返回；返回值类型为iterator。

4) end函数：const成员函数；调用红黑树的end函数并返回；返回值类型为const\_iterator。

5) rbegin函数：利用end函数构造reverse\_iterator；返回值类型为reverse\_iterator。

6) rbegin函数：const成员函数；利用end函数构造reverse\_iterator；返回值类型为const\_reverse\_iterator。

7) rend函数：利用begin函数构造reverse\_iterator；返回值类型为reverse\_iterator。

8) rend函数：const成员函数；利用begin函数构造reverse\_iterator；返回值类型为const\_reverse\_iterator。

9) cbegin函数：const成员函数；调用begin函数并返回；返回值类型为const\_iterator。

10) cend函数：const成员函数；调用end函数并返回；返回值类型为const\_iterator。

11) crbegin函数：const成员函数；调用rbegin函数并返回；返回值类型为const\_reverse\_iterator。

12) crend函数：const成员函数；调用rend函数并返回；返回值类型为const\_reverse\_iterator。

6. 容量相关函数

1) empty函数：const成员函数；调用红黑树的empty函数并返回；返回值类型为bool。

2) size函数：const成员函数；调用红黑树的size函数并返回；返回值类型为size\_type。

3) max\_size函数：const成员函数；调用红黑树的max\_size函数并返回；返回值类型为size\_type。

4) swap函数：接受一个参数rhs；调用红黑树的swap函数。

1. 插入删除相关函数
2. insert函数：接受一个参数x(value\_type)；调用红黑树的insert\_equal函数进行插入并返回；返回值类型为iterator。
3. insert函数：接受两个参数position和x；调用红黑树的insert\_equal函数进行插入并返回；返回值类型为iterator。
4. insert函数：模板函数，模板参数为InputIterator；接受两个参数first和last；调用红黑树的insert\_equal函数进行插入。
5. erase函数：接受一个参数position；调用红黑树的erase函数进行消除。
6. erase函数：接受一个参数key(key\_type)；调用红黑树的erase函数并返回；返回值类型为size\_type。
7. erase函数：接受两个参数first和last；调用红黑树的erase函数进行清除。
8. clear函数：调用红黑树的clear函数进行清除。
9. multiset相关操作函数
10. find函数：const成员函数；接受一个参数key(key\_type)；调用红黑树的find函数并返回；返回值类型为iterator。
11. count函数：const成员函数；接受一个参数key(key\_type)；调用红黑树的count函数并返回；返回值类型为size\_type。
12. lower\_bound函数：const成员函数；接受一个参数key(key\_type)；调用红黑树的lower\_bound函数并返回；返回值类型为iterator。
13. upper\_bound函数：const成员函数；接受一个参数key(key\_type)；调用红黑树的upper\_bound函数并返回；返回值类型为iterator。
14. equal\_range函数：const成员函数；接受一个参数key(key\_type)；调用红黑树的equal\_range函数并返回；返回值类型为pair<iterator，iterator>。
15. 友元函数
16. operator==：接受两个参数lhs和rhs；比较lhs和rhs的红黑树是否相等。
17. operator<：接受两个参数lhs和rhs；比较lhs的红黑树是否小于rhs的红黑树。
18. 其他函数
19. operator==：接受两个参数rhs和lhs；比较rhs和lhs是否相等。
20. operator<：接受两个参数rhs和lhs；比较lhs是否小于rhs。
21. operator!=：接受两个参数rhs和lhs；调用operator==。
22. operator>：接受两个参数rhs和lhs；调用operator<。
23. operator<=：接受两个参数rhs和lhs；调用operator<。
24. operator>=：接受两个参数rhs和lhs；调用operator<。
25. swap函数：接受两个参数rhs和lhs；利用lhs调用swap函数。

### map

1. map

模板类map，键值不允许重复，map中的元素都是pair；有四个模板参数Key，T，Compare(默认为less)和Alloc；typedef包括key\_type，data\_type，mapped\_type，value\_type(pair<const key, T>)和key\_compare；包含一个仿函数value\_compare，继承自binary\_function<value\_type，value\_type，bool>，仿函数中包含友元类map，protected属性包括一个比较函数comp和构造函数(传入比较函数)，public属性下包括重载()运算符，用于比较两个value\_type变量的大小；模板成员包括一个底层为红黑树的树tree。

2. typedef

包括pointer，const\_pointer，reference，const\_reference，iterator，const\_iterator，reverse\_iterator，const\_reverse\_iterator，size\_type，difference\_type，allocator\_types；全部都是通过底层红黑树定义的型别。

1. 构造和复制函数
2. 默认构造函数：将比较函数Compare传给红黑树进行构造。
3. 构造函数(声明为explicit)：接受一个参数comp；将comp传给红黑树进行构造。
4. 模板构造函数：有一个模板参数InputIterator；接受两个参数first和last；调用红黑树的默认构造函数，然后调用红黑树的insert\_unique函数进行构造。
5. 模板构造函数：有一个模板参数InputIterator；接受三个参数first，last和comp；调用红黑树的构造函数，然后调用红黑树的insert\_unique函数进行构造。
6. 构造函数：接受一个参数ilist(initializer\_list)；调用红黑树的默认构造函数，然后调用红黑树的insert\_unique函数进行构造。
7. 复制构造函数：接受一个参数rhs；调用红黑树的复制构造函数。
8. operator=：接受一个参数rhs；将rhs的红黑树赋给当前容器红黑树，返回\*this；返回值类型为map&。
9. operator=：接受一个参数ilist(initializer\_list)；先调用clear函数将当前红黑树清除，然后调用insert\_unique函数进行构造；返回值类型为map&。
10. 接口函数
11. key\_comp函数：const成员函数；调用红黑树的key\_comp并返回，返回值类型为key\_comp。
12. value\_comp函数：const成员函数；调用仿函数value\_compare并返回，返回值类型为value\_comp。
13. get\_allocator函数：const成员函数；调用红黑树的get\_allocator函数并返回；返回值类型为allocator\_type。
14. 迭代器相关函数
15. begin函数：调用红黑树的begin函数并返回；返回值类型为iterator。
16. begin函数：const成员函数；调用红黑树的begin函数并返回；返回值类型为const\_iterator。
17. end函数：调用红黑树的end函数并返回；返回值类型为iterator。
18. end函数：const成员函数；调用红黑树的end函数并返回；返回值类型为const\_iterator。
19. rbegin函数：利用end函数构造reverse\_iterator；返回值类型为reverse\_iterator。
20. rbegin函数：const成员函数；利用end函数构造reverse\_iterator；返回值类型为const\_reverse\_iterator。
21. rend函数：利用begin函数构造reverse\_iterator；返回值类型为reverse\_iterator。
22. rend函数：const成员函数；利用begin函数构造reverse\_iterator；返回值类型为const\_reverse\_iterator。
23. cbegin函数：const成员函数；调用begin函数并返回；返回值类型为const\_iterator。
24. cend函数：const成员函数；调用end函数并返回；返回值类型为const\_iterator。
25. crbegin函数：const成员函数；调用rbegin函数并返回；返回值类型为const\_reverse\_iterator。
26. crend函数：const成员函数；调用rend函数并返回；返回值类型为const\_reverse\_iterator。
27. 容量相关函数
28. empty函数：const成员函数；调用红黑树的empty函数并返回；返回值类型为bool。
29. size函数：const成员函数；调用红黑树的size函数并返回；返回值类型为size\_type。
30. max\_size函数：const成员函数；调用红黑树的max\_size函数并返回；返回值类型为size\_type。
31. swap函数：接受一个参数rhs；调用红黑树的swap函数。
32. 插入删除相关函数
33. insert函数：接受一个参数value；调用红黑树的insert\_unique函数并返回；返回值类型为pair<iterator，bool>。
34. insert函数：接受两个参数position和value；调用红黑树的insert\_unique函数并返回；返回值类型为iterator。
35. insert函数：模板函数，模板参数为InputIterator；接受两个参数first和last；调用红黑树的insert\_unique函数。
36. erase函数：接受一个参数position；调用红黑树的erase函数。
37. erase函数：接受一个参数key；调用红黑树的erase函数并返回；返回值类型为size\_type。
38. erase函数：接受两个参数first和last；调用红黑树的erase函数。
39. clear函数：调用红黑树的clear函数。
40. map相关函数
41. find函数：接受一个参数key；调用红黑树的find函数并返回；返回值类型为iterator。
42. find函数：const成员函数；接受一个参数key；调用红黑树的find函数并返回；返回值类型为const\_iterator。
43. count函数：接受一个参数key；调用红黑树的find函数并判断返回值是否等于end，如果等于返回0否则返回1；返回值类型为size\_type。
44. lower\_bound函数：接受一个参数key；调用红黑树的lower\_bound函数并返回；返回值类型为iterator。
45. lower\_bound函数：const成员函数；接受一个参数key；调用红黑树的lower\_bound函数并返回；返回值类型为const\_iterator。
46. upper\_bound函数：接受一个参数key；调用红黑树的upper\_bound函数并返回；返回值类型为iterator。
47. upper\_bound函数：const成员函数；接受一个参数key；调用红黑树的upper\_bound函数并返回；返回值类型为const\_iterator。
48. equal\_range函数：接受一个参数key；调用红黑树的equal\_range函数并返回；返回值类型为pair<iterator，iterator>。
49. equal\_range函数：const成员函数；接受一个参数key；调用红黑树的equal\_range函数并返回；返回值类型为pair<const\_iterator，const\_iterator>。
50. 访问元素相关函数
51. at函数：接受一个参数key；调用lower\_bound函数得到一个迭代器it，如果it等于end或者key小于it的key值，抛出out\_of\_range异常；否则返回it的data值；返回值类型为data\_type&。
52. at函数：和前面的at函数类型；是const成员函数；返回值类型为const data\_type&。
53. operator[]：接受一个参数key；调用lower\_bound函数得到一个迭代器it，如果it等于end或者it的key小于it的key值，调用insert函数进行插入，返回it的data值；返回值类型为data\_type&。
54. 友元函数
55. operator==：接受两个参数lhs和rhs；比较lhs和rhs的红黑树是否相等。
56. operator<：接受两个参数lhs和rhs；比较lhs的红黑树是否小于rhs的红黑树。
57. 其他函数
58. operator==：接受两个参数rhs和lhs；比较rhs和lhs是否相等。
59. operator<：接受两个参数rhs和lhs；比较lhs是否小于rhs。
60. operator!=：接受两个参数rhs和lhs；调用operator==。
61. operator>：接受两个参数rhs和lhs；调用operator<。
62. operator<=：接受两个参数rhs和lhs；调用operator<。
63. operator>=：接受两个参数rhs和lhs；调用operator<。
64. swap函数：接受两个参数rhs和lhs；利用lhs调用swap函数。

### multimap

1. multimap

模板类multimap，键值不允许重复，multimap中的元素都是pair；有四个模板参数Key，T，Compare(默认为less)和Alloc；typedef包括key\_type，data\_type，mapped\_type，value\_type(pair<const key, T>)和key\_compare；包含一个仿函数value\_compare，继承自unary\_function<value\_type，value\_type，bool>，仿函数中包含友元类multimap，protected属性包括一个比较函数comp和构造函数(传入比较函数)，public属性下包括重载()运算符，用于比较两个value\_type变量的大小；模板成员包括一个底层为红黑树的树tree。

1. typedef

包括pointer，const\_pointer，reference，const\_reference，iterator，const\_iterator，reverse\_iterator，const\_reverse\_iterator，size\_type，difference\_type，allocator\_types；全部都是通过底层红黑树定义的型别。

1. 构造和复制函数
2. 默认构造函数：将比较函数Compare传给红黑树进行构造。
3. 构造函数(声明为explicit)：接受一个参数comp；将comp传给红黑树进行构造。
4. 模板构造函数：有一个模板参数InputIterator；接受两个参数first和last；调用红黑树的默认构造函数，然后调用红黑树的insert\_equal函数进行构造。
5. 模板构造函数：有一个模板参数InputIterator；接受三个参数first，last和comp；调用红黑树的构造函数，然后调用红黑树的insert\_equal函数进行构造。
6. 构造函数：接受一个参数ilist(initializer\_list)；调用红黑树的默认构造函数，然后调用红黑树的insert\_equal函数进行构造。
7. 复制构造函数：接受一个参数rhs；调用红黑树的复制构造函数。
8. operator=：接受一个参数rhs；将rhs的红黑树赋给当前容器红黑树，返回\*this；返回值类型为map&。
9. operator=：接受一个参数ilist(initializer\_list)；先调用clear函数将当前红黑树清除，然后调用insert\_equal函数进行构造；返回值类型为map&。
10. 接口函数
11. key\_comp函数：const成员函数；调用红黑树的key\_comp并返回，返回值类型为key\_comp。
12. value\_comp函数：const成员函数；调用仿函数value\_compare并返回，返回值类型为value\_comp。
13. get\_allocator函数：const成员函数；调用红黑树的get\_allocator函数并返回；返回值类型为allocator\_type。
14. 迭代器相关函数
15. begin函数：调用红黑树的begin函数并返回；返回值类型为iterator。
16. begin函数：const成员函数；调用红黑树的begin函数并返回；返回值类型为const\_iterator。
17. end函数：调用红黑树的end函数并返回；返回值类型为iterator。
18. end函数：const成员函数；调用红黑树的end函数并返回；返回值类型为const\_iterator。
19. rbegin函数：利用end函数构造reverse\_iterator；返回值类型为reverse\_iterator。
20. rbegin函数：const成员函数；利用end函数构造reverse\_iterator；返回值类型为const\_reverse\_iterator。
21. rend函数：利用begin函数构造reverse\_iterator；返回值类型为reverse\_iterator。
22. rend函数：const成员函数；利用begin函数构造reverse\_iterator；返回值类型为const\_reverse\_iterator。
23. cbegin函数：const成员函数；调用begin函数并返回；返回值类型为const\_iterator。
24. cend函数：const成员函数；调用end函数并返回；返回值类型为const\_iterator。
25. crbegin函数：const成员函数；调用rbegin函数并返回；返回值类型为const\_reverse\_iterator。
26. crend函数：const成员函数；调用rend函数并返回；返回值类型为const\_reverse\_iterator。
27. 容量相关函数
28. empty函数：const成员函数；调用红黑树的empty函数并返回；返回值类型为bool。
29. size函数：const成员函数；调用红黑树的size函数并返回；返回值类型为size\_type。
30. max\_size函数：const成员函数；调用红黑树的max\_size函数并返回；返回值类型为size\_type。
31. swap函数：接受一个参数rhs；调用红黑树的swap函数。
32. 插入删除相关函数
33. insert函数：接受一个参数value；调用红黑树的insert\_equal函数并返回；返回值类型为iterator。
34. insert函数：接受两个参数position和value；调用红黑树的insert\_equal函数并返回；返回值类型为iterator。
35. insert函数：模板函数，模板参数为InputIterator；接受两个参数first和last；调用红黑树的insert\_equal函数。
36. erase函数：接受一个参数position；调用红黑树的erase函数。
37. erase函数：接受一个参数key；调用红黑树的erase函数并返回；返回值类型为size\_type。
38. erase函数：接受两个参数first和last；调用红黑树的erase函数。
39. clear函数：调用红黑树的clear函数。
40. map相关函数
41. find函数：接受一个参数key；调用红黑树的find函数并返回；返回值类型为iterator。
42. find函数：const成员函数；接受一个参数key；调用红黑树的find函数并返回；返回值类型为const\_iterator。
43. count函数：接受一个参数key；调用红黑树的count函数并返回；返回值类型为size\_type。
44. lower\_bound函数：接受一个参数key；调用红黑树的lower\_bound函数并返回；返回值类型为iterator。
45. lower\_bound函数：const成员函数；接受一个参数key；调用红黑树的lower\_bound函数并返回；返回值类型为const\_iterator。
46. upper\_bound函数：接受一个参数key；调用红黑树的upper\_bound函数并返回；返回值类型为iterator。
47. upper\_bound函数：const成员函数；接受一个参数key；调用红黑树的upper\_bound函数并返回；返回值类型为const\_iterator。
48. equal\_range函数：接受一个参数key；调用红黑树的equal\_range函数并返回；返回值类型为pair<iterator，iterator>。
49. equal\_range函数：const成员函数；接受一个参数key；调用红黑树的equal\_range函数并返回；返回值类型为pair<const\_iterator，const\_iterator>。
50. 友元函数
51. operator==：接受两个参数lhs和rhs；比较lhs和rhs的红黑树是否相等。
52. operator<：接受两个参数lhs和rhs；比较lhs的红黑树是否小于rhs的红黑树。
53. 其他函数
54. operator==：接受两个参数rhs和lhs；比较rhs和lhs是否相等。
55. operator<：接受两个参数rhs和lhs；比较lhs是否小于rhs。
56. operator!=：接受两个参数rhs和lhs；调用operator==。
57. operator>：接受两个参数rhs和lhs；调用operator<。
58. operator<=：接受两个参数rhs和lhs；调用operator<。
59. operator>=：接受两个参数rhs和lhs；调用operator<。
60. swap函数：接受两个参数rhs和lhs；利用lhs调用swap函数。

### hashtable

hashtable使用开链法实现。

1. hashtable结点

是一个模板类，有一个模板参数T；包含两个成员next和value；包含默认构造函数，接受一个参数n的构造函数和一个复制构造函数。

1. hashtable迭代器(base版本)

是一个模板类，有六个模板参数Value，Key，HashFcn，ExtractKey，EqualKey和Alloc；继承iterator(类型为forward\_iterator)；typedef包括hashtable，base\_iterator，iterator，const\_iterator，node，node\_ptr，container\_ptr，const\_node\_ptr，const\_container\_ptr，size\_type和difference\_type；包含两个成员node(node\_ptr)和ht(container\_ptr)；有一个默认构造函数；重载operator==和operator!=，判断两个迭代器的node是否相等。

1. hashtable迭代器

是一个模板类，有六个模板参数Value，Key，HashFcn，ExtractKey，EqualKey和Alloc；继承base\_iterator；typedef包括base\_iterator，hashtable，iterator，const\_iterator，node\_ptr，container\_ptr，value\_type，pointer，reference，size\_type和difference\_type；使用base\_iterator中的node和ht。

1. 构造和复制函数
2. 默认构造函数：直接使用default。
3. 构造函数：接受两个参数n和tab；将n赋值给node，tab赋值给ht。
4. 复制构造函数：接受一个参数rhs(iterator)；将rhs的node赋值给node，rhs的ht赋值给ht。
5. 复制构造函数：接受一个参数rhs(const\_iterator)；将rhs的node赋值给node，rhs的ht赋值给ht。
6. operator=：接受一个参数rhs(iterator)；如果rhs不等于this，将rhs的node和ht分别赋值给node和ht；返回\*this；返回值类型为iterator&。
7. operator=：接受一个参数rhs(const\_iterator)；如果rhs不等于this，将rhs的node和ht分别赋值给node和ht；返回\*this；返回值类型为iterator&。
8. 重载操作符
9. operator\*：const成员函数；返回node的value值；返回值类型为reference。
10. operator->：const成员函数；调用operator\*；返回值类型为pointer。
11. operator++：将当前node声明为old(const node\_ptr)，将node设为node->next，如果node为空，通过ht调用bkt\_num函数得到bucket编号bucket，利用while循环，当node不为空且++bucket小于当前ht的size，node等于当前bucket编号的第一个结点，返回\*this；返回值类型为iterator&。
12. operator++(int)：调用operator++；返回值类型为iterator。
13. hashtable迭代器(const\_iterator)

实现上和iterator类似，只是相应部分换成了const\_iterator。

1. hashtable桶个数

利用enum将stl\_num\_primes设为28；维护一个静态const unsigned long数组stl\_prime\_list {

  53ul,         97ul,         193ul,       389ul,       769ul,

  1543ul,       3079ul,       6151ul,      12289ul,     24593ul,

  49157ul,      98317ul,      196613ul,    393241ul,    786433ul,

  1572869ul,    3145739ul,    6291469ul,   12582917ul,  25165843ul,

  50331653ul,   100663319ul,  201326611ul, 402653189ul, 805306457ul, 1610612741ul, 3221225473ul, 4294967291ul

};

定义一个内联函数stl\_next\_prime用于得到最接近并大于等于n的质数；stl\_next\_prime中定义一个指针first指向stl\_prime\_list数组的开头，指针last指向stl\_prime\_list的最后，调用lower\_bound函数得到一个指针pos，如果pos等于last返回数组中最后一个元素，否则返回pos位置的元素。

1. hashtable

是一个模板类，包含六个模板参数Value，Key，HashFcn，ExtractKey，EqualKey和Alloc；将iterator和const\_iterator声明为友元类；typedef包括key\_type，value\_type，hasher，key\_equal，get\_key，node\_ptr，container\_ptr，allocator\_type，data\_allocator，node\_allocator，pointer，const\_pointer，reference，const\_reference，size\_type，difference\_type，iterator和const\_iterator；函数get\_allocator用于获得配置器；private属性中包含buckets(vector<node\_ptr>)，equal，hash，get\_key，bucket\_size和num\_elements，get\_node函数通过node\_allocator的allocate函数得到一个结点，返回值类型为node\_ptr，put\_node函数通过node\_allocator的deallocate函数销毁一个结点(接受一个参数p)；hash\_funct函数返回hash函数，key\_eq函数返回equal函数。

1. 构造和复制函数
2. 构造函数：接受四个参数n，hf，eql和ext；将hf，eql和ext分别赋值给hash，equal和get\_key，将num\_elements设为0；调用辅助函数initialize\_buckets进行初始化。
3. 构造函数：接受三个参数n，hf和eql；将hf和eql分别赋值给hash和equal，将ExtractKey临时对象赋值给ext，将num\_elements设为0；调用辅助函数initialize\_buckets进行初始化。
4. 复制构造函数：接受一个参数rhs；将rhs的hash，equal和get\_key分别赋值给hash，equal和get\_key；调用辅助函数copy\_from。
5. operator=：接受一个参数rhs；如果rhs不等于this，调用clear函数，将rhs的hash，equal和get\_key分别赋值给hash，equal和get\_key，调用copy\_from函数，返回\*this；返回值类型为hashtable&。
6. 析构函数：调用clear函数。
7. 迭代器相关操作
8. begin函数：利用for循环寻找第一个不为空的桶，如果找到利用桶中第一个结点和当前hashtable构造一个迭代器并返回；返回值类型为iterator。
9. begin函数：const成员函数；和前一个begin函数类似，只是构造的迭代器类型为const\_iterator；返回值类型为const\_iterator。
10. end函数：利用空指针和当前hashtable构造一个迭代器并返回；返回值类型为iterator。
11. end函数：const成员函数；和前一个end函数类似，只是构造的迭代器类型为const\_iterator；返回值类型为const\_iterator。
12. cbegin函数：const成员函数；调用begin函数；返回值类型为const\_iterator。
13. cend函数：const成员函数；调用end函数；返回值类型为const\_iterator。
14. 容量相关函数
15. empty函数：const成员函数；判断num\_elements是否为0；返回值类型为bool。
16. size函数：const成员函数；返回num\_elements的值；返回值类型为size\_type。
17. max\_size函数：const成员函数；返回size\_type下-1的值；返回值类型为size\_type。
18. bucket\_count函数：const成员函数；通过buckets调用size函数；返回值类型为size\_type。
19. max\_bucket\_count函数：const成员函数；返回stl\_primelist中的最后一个数；返回值类型为size\_type。
20. elems\_in\_bucket函数：const成员函数；接受一个参数bucket；利用for循环遍历桶中所有结点并返回个数；返回值类型为size\_type。
21. swap函数：接受一个参数rhs；如果this不等于rhs，通过buckets调用swap函数进行交换，调用mystl的swap函数将hash，equal，get\_key和num\_elements交换。
22. 插入删除相关函数
23. insert\_unique函数：接受一个参数value；调用resize函数重新分配空间，调用insert\_unique\_noresize函数并返回；返回值类型为pair<iterator，bool>。
24. insert\_unique\_noresize函数：接受一个参数value；首先调用bkt\_num函数计算对应的桶的编号n，得到对应桶的第一个结点first；利用for循环遍历桶中全部结点，如果已经存在key值相等的结点，将当前结点和hashtable声明为一个迭代器并打包false一起返回；否则调用new\_node函数创建一个新的结点，利用头插法将temp插入到桶的头部，将num\_elements加一，最后返回；返回值类型为pair<iterator，bool>。
25. insert\_equal函数：接受一个参数value；调用resize函数重新分配空间，调用insert\_equal\_noresize函数并返回；返回值类型为iterator。
26. insert\_equal\_noresize函数：接受一个参数value；首先调用bkt\_num函数计算对应的桶的编号n，得到对应桶的第一个结点first；利用for循环遍历桶中全部结点，如果已经存在key值相等的结点，调用new\_node函数获得一个新的结点temp，将temp插入相等结点cur之后，将num\_elements加一并返回；否则调用new\_node函数创建一个新的结点temp，利用头插法将temp插入到桶的头部，将num\_elements加一并返回；返回值类型为iterator。
27. insert\_unique函数：模板函数，有一个模板参数InputIterator；接受两个参数first和last；根据传入迭代器的类型调用函数insert\_unique。
28. insert\_unique函数(InputIterator版本)：模板函数，有一个模板参数InputIterator；接受三个参数first，last和input\_iterator\_tag；利用for循环依次调用insert\_unique函数。
29. insert\_unique函数(ForwardIterator版本)：模板函数，有一个模板参数ForwardIterator；接受三个参数first，last和forward\_iterator\_tag；首先调用distance函数算出first和last之间的距离n，调用resize函数重新分配空间，利用for循环(这里用的是n--)依次调用insert\_unique\_noresize函数。
30. insert\_equal函数：模板函数，有一个模板参数InputIterator；接受两个参数first和last；根据传入迭代器的类型调用函数insert\_equal。
31. insert\_equal函数(InputIterator版本)：模板函数，有一个模板参数InputIterator；接受三个参数first，last和input\_iterator\_tag；利用for循环依次调用insert\_equal函数。
32. insert\_equal函数(ForwardIterator版本)：模板函数，有一个模板参数ForwardIterator；接受三个参数first，last和forward\_iterator\_tag；首先调用distance函数算出first和last之间的距离n，调用resize函数重新分配空间，利用for循环(这里用的是n--)依次调用insert\_equal\_noresize函数。
33. insert\_unique函数：接受两个参数first和last(const value\_type\*)；首先算出first和last之间的距离n，调用resize函数重新分配空间，利用for循环(这里用的是n--)依次调用insert\_unique\_noresize函数。
34. insert\_equal函数：接受两个参数first和last(const value\_type\*)；首先算出first和last之间的距离n，调用resize函数重新分配空间，利用for循环(这里用的是n--)依次调用insert\_equal\_noresize函数。
35. insert\_unique函数：接受两个参数first和last(const\_iterator)；首先调用distance函数算出first和last之间的距离n，调用resize函数重新分配空间，利用for循环(这里用的是n--)依次调用insert\_unique\_noresize函数。
36. insert\_equal函数：接受两个参数first和last(const\_iterator)；首先调用distance函数算出first和last之间的距离n，调用resize函数重新分配空间，利用for循环(这里用的是n--)依次调用insert\_equal\_noresize函数。
37. erase函数：接受一个参数key；调用bkt\_num\_key函数获得桶的编号n，得到桶中第一个结点first，将要删除的结点个数erased设为0；如果first不为空，将当前结点设为cur，下一结点设为next，利用while循环，当next不为空时，如果next结点的key值等于key，修改当前结点的next指针，调用delete\_node函数删除next结点，调整next结点，将erased++，num\_elements--，否则向下遍历；最后判断first结点的key值是否等于key，如果是进行删除；最后返回erased；返回值类型为size\_type。
38. erase函数：接受一个参数it；首先获得it的结点p，如果p不为空，通过bkt\_num函数获得桶的编号n，获得桶中第一个结点cur，如果cur等于p，则将cur删除并做相应调整；否则声明一个next结点，利用while循环，当next不为空时，如果next等于p，将next结点删除并做相应调整(记得删除后break退出)，否则继续向下遍历。
39. erase函数：接受一个参数it(const\_iterator)；利用const\_cast将it的结点和哈希表去掉const属性，然后重新得到一个迭代器，调用erase函数。
40. erase函数：接受两个参数first和last；首先获得桶的编号first\_bucket(如果first的结点不为空则调用bkt\_num函数，否则编号为bucket的size)，同理获得桶的编号last\_bucket；如果first的结点等于last的结点，直接返回；否则如果first\_bucket等于last\_bucket，调用erase\_bucket函数(三个参数)，否则调用rease\_bucket函数(两个参数)将first到桶末端之间的结点全部删除，然后利用for循环依次调用erase\_bucket函数进行结点的删除，如果last\_bucket不等于bucket的size，调用erase\_bucket函数将last\_bucket到last结点之间的结点删除。
41. erase函数：接受两个参数first和last(const\_iterator)；利用const\_cast将first和last的结点和哈希表去掉const属性，然后重新得到两个迭代器，调用erase函数。
42. clear函数：利用for循环遍历所有的桶，获得桶中第一个结点cur，如果cur不为空，利用while循环删除桶中所有结点，将桶的第一个结点设为空，将num\_elements设为0。
43. 查找相关函数
44. find\_or\_insert：接受一个参数value；首先调用resize重新调整空间，然后算出桶的编号n和桶中第一个结点first，利用for循环遍历桶中所有结点，如果找到返回对应的值，否则利用new\_node函数创建一个新的结点temp，然后将temp插入桶的头部并做相应调整，返回插入的值；返回值类型为reference。
45. find函数：接受一个参数key；首先算出桶对应的编号n和桶中第一个结点first，利用for循环遍历桶中所有结点，如果找到退出循环，将first和当前hashtable声明为一个迭代器并返回；返回值类型为iterator。
46. find函数：和上述find函数类似，变成了const版本。
47. count函数：const成员函数；接受一个参数key；首先算出桶的编号n，利用for循环遍历桶中所有结点并找出key值等于key的结点，最后将结果返回；返回值类型为size\_type。
48. equal\_range函数：接受一个参数key；首先计算出桶的编号n，利用for循环遍历桶中所有结点，如果找到一个key值等于key的结点，继续利用for循环遍历，直到当前结点的key值不等于key，利用两个结点构造两个迭代器并返回；如果后面结点的key值全部等于key，利用for循环找到下一个非空的桶的第一个结点，构造两个迭代器并返回；如果没有一个结点的key值等于key，返回两个end构造的迭代器；返回值类型为pair<iterator，iterator>。
49. equal\_range函数：和上述equal\_range函数类似，变成了const版本。
50. 辅助函数
51. next\_size函数：const成员函数；接受一个参数n；调用stl\_next\_prime函数并返回；返回值类型为size\_type。
52. initialize\_buckets函数：接受一个参数n；首先调用next\_size函数得到需要的桶的个数n\_buckets，然后通过buckets调用reserve函数分配空间，通过buckets调用assign函数将全部桶设为空，将num\_elements设为0。
53. copy\_from函数：接受一个参数ht；首先通过bucket调用clear函数将bucket清空，然后通过bucket调用reserve函数分配空间，通过bucket函数调用assign函数将全部桶设为空；利用for循环遍历ht的全部桶，如果当前桶中有结点，先调用new\_node函数得到一个结点copy，然后将copy放入当前桶中，再利用for循环将整个桶的结点进行复制，将num\_elements赋值为ht的num\_elements；如果空间分配失败调用clear函数(这里使用try和catch实现)。
54. new\_node函数：接受一个参数value；首先调用get\_node函数获得一个新的结点，将新结点的next设为空，利用mystl的constrcut函数进行构造，返回新结点；如果失败调用put\_node函数(这里使用try和catch实现)。
55. delete\_node函数：接受一个参数node；首先调用mystl的destroy对value进行清除，然后调用put\_node函数。
56. resize函数：接受一个参数n；首先通过bucket调用size函数得到原来桶的个数old\_n；如果n大于old\_n，调用next\_size函数得到需要的桶的个数new\_n；如果new\_n大于old\_n，构造一个大小为new\_n的新的bucket数组temp，利用for循环遍历非空的桶(这里就是将桶中所有结点重新计算在新的数组中的位置并放置)，对于非空的桶，利用while循环，调用bkt\_num函数计算出该桶在新的数组中的编号new\_bucket，然后将桶链接到下一个结点(first->next)，利用头插法将first结点插入到新的桶中，更新first的值；调用bucket的swap函数将当前数组和temp进行交换；如果发生异常，利用for循环和while循环将temp中的所有非空桶的所有结点调用delete\_node函数进行清除，将异常继续传播(这里使用try和catch实现)。
57. bkt\_num函数：const成员函数；接受一个参数value；使用get\_key获得value对应的key值，然后调用bkt\_num\_key函数并返回；返回值类型为size\_type。
58. bkt\_num函数：const成员函数；接受两个参数value和n；使用get\_key获得value对应的key值，然后调用bkt\_num\_key函数并返回；返回值类型为size\_type。
59. bkt\_num\_key函数：const成员函数；接受一个函数key；调用函数bkt\_num\_key并返回(将bucket的size作为第二个参数传入)；返回值类型为size\_type。
60. bkt\_num\_key函数：const成员函数；接受两个参数key和n；将key传入hash函数并将返回值模n；返回值类型为size\_type。
61. erase\_bucket函数：接受三个参数n，first和last(node\_ptr)；得到桶中第一个结点cur，如果cur等于first，调用erase\_bucket函数进行删除；否则声明一个next结点，利用for循环遍历，直到next等于first，利用while循环，当next不等于last时，删除next结点并做相应调整。
62. erase\_bucket函数：接受两个参数n和last(node\_ptr)；首先获得桶中第一个结点cur，利用while循环，当cur不等于last时，删除cur结点并做相应调整(这里要不断地设置桶的第一个结点)。
63. 其他函数
64. operator==：接受两个参数lhs和rhs；如果两个哈希表桶的数量不相等，返回false；利用for循环比较每个桶的结点，如果有不相等返回false，否则返回true；返回值类型为bool。
65. operator!=：接受两个参数lhs和rhs；调用operator==。
66. swap函数：接受两个参数lhs和rhs；调用lhs的swap函数。

### unordered\_set

1. hash\_function

哈希函数由仿函数hash实现，hash类中什么都不用实现，针对不同的数据类型做不同的全特化。

1. stl\_hash\_string函数(内联函数)：接受一个参数s(const char\*)；利用for循环，将之前得到的数\*5再加上当前字符；返回值类型为size\_t。
2. char\*全特化：重载operator()(const函数，参数类型为const char\*，返回值类型为size\_t)，调用stl\_hash\_string函数并返回。
3. const char\*全特化：重载operator()(const函数，参数类型为const char\*，返回值类型为size\_t)，调用stl\_hash\_string函数并返回。
4. char全特化：重载operator()(const函数，返回值类型为size\_t)，直接将参数x返回。
5. unsigned char全特化：重载operator()(const函数，返回值类型为size\_t)，直接将参数x返回。
6. signed char全特化：重载operator()(const函数，返回值类型为size\_t)，直接将参数x返回。
7. short全特化：重载operator()(const函数，返回值类型为size\_t)，直接将参数x返回。
8. unsigned short全特化：重载operator()(const函数，返回值类型为size\_t)，直接将参数x返回。
9. int全特化：重载operator()(const函数，返回值类型为size\_t)，直接将参数x返回。
10. unsigned int全特化：重载operator()(const函数，返回值类型为size\_t)，直接将参数x返回。
11. long全特化：重载operator()(const函数，返回值类型为size\_t)，直接将参数x返回。
12. unsigned long全特化：重载operator()(const函数，返回值类型为size\_t)，直接将参数x返回。
13. bool全特化：重载operator()(const函数，返回值类型为size\_t)，直接将参数x返回。
14. wchar\_t全特化：重载operator()(const函数，返回值类型为size\_t)，直接将参数x返回。
15. char16\_t全特化：重载operator()(const函数，返回值类型为size\_t)，直接将参数x返回。
16. char32\_t全特化：重载operator()(const函数，返回值类型为size\_t)，直接将参数x返回。
17. long long全特化：重载operator()(const函数，返回值类型为size\_t)，直接将参数x返回。
18. unsigned long long全特化：重载operator()(const函数，返回值类型为size\_t)，直接将参数x返回。
19. 针对指针的偏特化版本：利用reinterpret\_cast将参数转化为size\_t类型并返回。
20. fnv\_hash(inline函数)：针对浮点数的hash函数；接受两个参数first(const unsigned char\*)和count(size\_t)；定义两个常量 fnv\_offset(14695981039346656037ull;) 和 fnv\_prime(1099511628211ull)，它们是用于计算哈希值的起始值和乘数；声明一个res等于0，利用for循环遍历输入数据的每个字节，对 result 进行异或和乘法操作，返回res；返回值类型为size\_t。
21. float全特化：重载operator()(const函数，返回值类型为size\_t)，判断val是否为0.0f，如果是返回0，否则调用fnv\_hash函数并返回。
22. double全特化：重载operator()(const函数，返回值类型为size\_t)，判断val是否为0.0f，如果是返回0，否则调用fnv\_hash函数并返回。
23. long double全特化：重载operator()(const函数，返回值类型为size\_t)，判断val是否为0.0f，如果是返回0，否则调用fnv\_hash函数并返回。
24. 万能的hash函数

根据特定的类，创建特定的hash函数，hash函数中重载operator()(const函数，参数类型为特定类，返回值为size\_t)，operator()调用函数hash\_val。

1. hash\_val函数(inlin函数)：模板函数，接受任意多个模板参数(使用typename…实现)；接受任意多个参数args(const Types&…)；声明一个size\_t变量seed，调用hash\_val函数，返回seed；返回值类型为size\_t。
2. hash\_val函数(inline函数)：模板参数，接受一个模板参数T和任意多个模板参数Types；接受三个参数seed，val(T)和args(const Types&…)；调用hash\_combine函数，调用hash\_val函数。
3. hash\_val函数(inline函数)：模板参数，接受一个模板参数T；接受两个参数seed和val；调用hash\_combine函数。
4. hash\_combine函数(inline函数)：模板参数，接受一个模板参数T；接受两个参数seed和val；调用类型T的hash函数对val进行hash，再加上0x9e3779b9，再加上(seed<<6)和(seed>>2)，结果和seed进行异或。
5. unordered\_set

模板类，有四个模板参数Value，HashFcn，EqualKey和Alloc；内部包含一个哈希表ht；typedef包括allocator\_type，key\_type，value\_type，hasher，key\_equal，size\_type，difference\_type，pointer，const\_pointer，reference，const\_reference，iterator和const\_iterator；包含一个get\_allocator函数(调用哈希表的get\_allocator函数)，hash\_funct调用哈希表的hash\_funct函数并返回，key\_eq函数调用哈希表的key\_eq函数并返回；将operator==声明为友元函数。

1. 构造和复制函数
2. 构造函数：初始化哈希表，将桶的个数设为100，将hasher和key\_equal的默认值传入进行构造。
3. 构造函数：声明为explicit；接受一个参数n；初始化哈希表，将桶的数量设为n，将hasher和key\_equal的默认值传入进行构造。
4. 构造函数：接受两个参数n和hf；初始化哈希表，将桶的数量设为n，哈希函数设为hf，key\_equal的默认值传入进行构造。
5. 构造函数：接受三个参数n，hf和eql；初始化哈希表，将三个参数传入进行构造。
6. 构造函数：模板函数，有一个模板参数InputIterator；接受两个参数first和last；默认的构造哈希表，调用哈希表的insert\_unique函数进行插入构造。
7. 构造函数：模板函数，有一个模板参数InputIterator；接受三个参数first，last和n；构造哈希表，调用哈希表的insert\_unique函数进行插入构造。
8. 构造函数：模板函数，有一个模板参数InputIterator；接受四个参数first，last，n和hf；构造哈希表，调用哈希表的insert\_unique函数进行插入构造。
9. 构造函数：模板函数，有一个模板参数InputIterator；接受五个参数first，last，n，hf和eql；默认的构造哈希表，调用哈希表的insert\_unique函数进行插入构造。
10. 构造函数：同样的方式只是模板参数改为了initializer\_list。
11. 复制构造函数：接受一个参数rhs；将rhs的哈希表赋给当前容器。
12. operator=：接受一个参数rhs；如果this不等于rhs，将rhs的哈希表赋给当前容器，返回this；返回值类型为unordered\_set&。
13. operator=：接受一个参数rhs(initializer\_list)；调用哈希表的clear函数，调用哈希表的reserve函数，调用哈希表的insert\_unique函数，返回this；返回值类型为unordered\_set&。
14. 析构函数：采用默认方式(default)。
15. 迭代器相关函数
16. begin函数：调用哈希表的begin函数；返回值类型为iterator。
17. begin函数：const成员函数；调用哈希表的begin函数；返回值类型为const\_iterator。
18. end函数：调用哈希表的end函数；返回值类型为iterator。
19. end函数：const成员函数；调用哈希表的end函数；返回值类型为const\_iterator。
20. cbegin函数：const成员函数；调用哈希表的cbegin函数；返回值类型为const\_iterator。
21. cend函数：const成员函数；调用哈希表的cend函数；返回值类型为const\_iterator。
22. 容量相关函数
23. empty函数：const成员函数；调用哈希表的empty函数，返回值类型为bool。
24. size函数：const成员函数；调用哈希表的size函数，返回值类型为size\_type。
25. max\_size函数：const成员函数；调用哈希表的max\_size函数；返回值类型为size\_type。
26. resize函数：接受一个参数n；调用哈希表的resize函数。
27. bucket\_count函数：const成员函数；调用哈希表的bucket\_count函数并返回；返回值类型为size\_type。
28. max\_bucket\_count函数：const成员函数；调用哈希表的max\_bucket\_count函数并返回；返回值类型为size\_type。
29. elems\_in\_bucket函数：const成员函数；接受一个参数bucket；调用哈希表的elems\_in\_bucket函数并返回；返回值类型为size\_type。
30. swap函数：接受一个参数rhs；调用哈希表的swap函数。
31. 插入删除相关函数
32. insert函数：接受一个参数value；调用哈希表的insert\_unique函数并返回，返回值类型为pair<iterator，bool>。
33. insert函数：模板函数，有一个模板参数InputIterator；接受两个参数first和last；调用哈希表的insert\_unique函数。
34. insert函数：接受两个参数first和last(const value\_type&)；调用哈希表的insert\_unique函数。
35. insert函数：接受两个参数first和last(const\_iterator)；调用哈希表的insert\_unique函数。
36. insert\_noresize函数：接受一个参数value；调用哈希表的insert\_unique函数并返回；返回值类型为pair<iterator，bool>。
37. erase函数：接受一个参数key；调用哈希表的erase函数并返回；返回值类型为size\_type。
38. erase函数：接受一个参数it；调用哈希表的erase函数。
39. erase函数：接受两个参数first和last；调用哈希表的erase函数。
40. clear函数：调用哈希表的clear函数。
    1. 查找相关函数
41. find函数：const成员函数；接受一个参数key；调用哈希表的find函数并返回；返回值类型为iterator。
42. count函数：const成员函数；接受一个参数key；调用哈希表的count函数并返回；返回值类型为size\_type。
43. equal\_range函数：const成员函数；接受一个参数key；调用哈希表的equal\_range函数并返回；返回值类型为pair<iterator，iterator>。
    1. 其他函数
44. operator==(inline函数)：接受两个参数lhs和rhs；判断lhs是否等于rhs。
45. operator!=(inline函数)：接受两个参数lhs和rhs；调用operator==。
46. swap函数(inline函数)：接受两个参数lhs和rhs；通过lhs调用swap函数。

### unordered\_multiset

1. unordered\_multiset

模板类，有四个模板参数Value，HashFcn，EqualKey和Alloc；内部包含一个哈希表ht；typedef包括allocator\_type，key\_type，value\_type，hasher，key\_equal，size\_type，difference\_type，pointer，const\_pointer，reference，const\_reference，iterator和const\_iterator；包含一个get\_allocator函数(调用哈希表的get\_allocator函数)，hash\_funct调用哈希表的hash\_funct函数并返回，key\_eq函数调用哈希表的key\_eq函数并返回；将operator==声明为友元函数。

1) 构造和复制函数

a) 构造函数：初始化哈希表，将桶的个数设为100，将hasher和key\_equal的默认值传入进行构造。

b) 构造函数：声明为explicit；接受一个参数n；初始化哈希表，将桶的数量设为n，将hasher和key\_equal的默认值传入进行构造。

c) 构造函数：接受两个参数n和hf；初始化哈希表，将桶的数量设为n，哈希函数设为hf，key\_equal的默认值传入进行构造。

d) 构造函数：接受三个参数n，hf和eql；初始化哈希表，将三个参数传入进行构造。

e) 构造函数：模板函数，有一个模板参数InputIterator；接受两个参数first和last；默认的构造哈希表，调用哈希表的insert\_equal函数进行插入构造。

f) 构造函数：模板函数，有一个模板参数InputIterator；接受三个参数first，last和n；构造哈希表，调用哈希表的insert\_equal函数进行插入构造。

g) 构造函数：模板函数，有一个模板参数InputIterator；接受四个参数first，last，n和hf；构造哈希表，调用哈希表的insert\_equal函数进行插入构造。

h) 构造函数：模板函数，有一个模板参数InputIterator；接受五个参数first，last，n，hf和eql；默认的构造哈希表，调用哈希表的insert\_equal函数进行插入构造。

i) 构造函数：同样的方式只是模板参数改为了initializer\_list。

j) 复制构造函数：接受一个参数rhs；将rhs的哈希表赋给当前容器。

k) operator=：接受一个参数rhs；如果this不等于rhs，将rhs的哈希表赋给当前容器，返回this；返回值类型为unordered\_set&。

l) operator=：接受一个参数rhs(initializer\_list)；调用哈希表的clear函数，调用哈希表的reserve函数，调用哈希表的insert\_unique函数，返回this；返回值类型为unordered\_set&。

m) 析构函数：采用默认方式(default)。

2) 迭代器相关函数

a) begin函数：调用哈希表的begin函数；返回值类型为iterator。

b) begin函数：const成员函数；调用哈希表的begin函数；返回值类型为const\_iterator。

c) end函数：调用哈希表的end函数；返回值类型为iterator。

d) end函数：const成员函数；调用哈希表的end函数；返回值类型为const\_iterator。

e) cbegin函数：const成员函数；调用哈希表的cbegin函数；返回值类型为const\_iterator。

f) cend函数：const成员函数；调用哈希表的cend函数；返回值类型为const\_iterator。

3) 容量相关函数

a) empty函数：const成员函数；调用哈希表的empty函数，返回值类型为bool。

b) size函数：const成员函数；调用哈希表的size函数，返回值类型为size\_type。

c) max\_size函数：const成员函数；调用哈希表的max\_size函数；返回值类型为size\_type。

d) resize函数：接受一个参数n；调用哈希表的resize函数。

e) bucket\_count函数：const成员函数；调用哈希表的bucket\_count函数并返回；返回值类型为size\_type。

f) max\_bucket\_count函数：const成员函数；调用哈希表的max\_bucket\_count函数并返回；返回值类型为size\_type。

g) elems\_in\_bucket函数：const成员函数；接受一个参数bucket；调用哈希表的elems\_in\_bucket函数并返回；返回值类型为size\_type。

h) swap函数：接受一个参数rhs；调用哈希表的swap函数。

4) 插入删除相关函数

a) insert函数：接受一个参数value；调用哈希表的insert\_equal函数并返回，返回值类型为iterator。

b) insert函数：模板函数，有一个模板参数InputIterator；接受两个参数first和last；调用哈希表的insert\_equal函数。

c) insert函数：接受两个参数first和last(const value\_type&)；调用哈希表的insert\_equal函数。

d) insert函数：接受两个参数first和last(const\_iterator)；调用哈希表的insert\_equal函数。

e) insert\_noresize函数：接受一个参数value；调用哈希表的insert\_equal函数并返回；返回值类型为iterator。

f) erase函数：接受一个参数key；调用哈希表的erase函数并返回；返回值类型为size\_type。

g) erase函数：接受一个参数it；调用哈希表的erase函数。

h) erase函数：接受两个参数first和last；调用哈希表的erase函数。

i) clear函数：调用哈希表的clear函数。

5) 查找相关函数

a) find函数：const成员函数；接受一个参数key；调用哈希表的find函数并返回；返回值类型为iterator。

b) count函数：const成员函数；接受一个参数key；调用哈希表的count函数并返回；返回值类型为size\_type。

c) equal\_range函数：const成员函数；接受一个参数key；调用哈希表的equal\_range函数并返回；返回值类型为pair<iterator，iterator>。

6) 其他函数

a) operator==(inline函数)：接受两个参数lhs和rhs；判断lhs是否等于rhs。

b) operator!=(inline函数)：接受两个参数lhs和rhs；调用operator==。

c) swap函数(inline函数)：接受两个参数lhs和rhs；通过lhs调用swap函数。

1. insert\_iterator偏特化
2. unordered\_set

模板类，有四个模板参数Key，HashFcn，EqualKey和Alloc；protected下维护一个容器指针；typedef包括container\_type，iterator\_category(output\_iterator\_tag)，value\_type，difference\_type，pointer和reference；构造函数接受一个参数x和两个参数x和iterator(类型)；重载operator=，接受一个参数value，调用容器的insert函数并返回自身；operator\*，operator++和operator++(int)都返回自身。

1. unordered\_multiset

和unordered\_set类似。

### unordered\_map

1. unordered\_map

模板类，有五个模板参数Key，T，HashFcn，EqualKey和Alloc；内部包含一个哈希表ht；typedef包括allocator\_type，key\_type，data\_type，mapped\_type，value\_type，hasher，key\_equal，size\_type，difference\_type，pointer，const\_pointer，reference，const\_reference，iterator和const\_iterator；包含一个get\_allocator函数(调用哈希表的get\_allocator函数)，hash\_funct调用哈希表的hash\_funct函数并返回，key\_eq函数调用哈希表的key\_eq函数并返回；将operator==声明为友元函数。

1) 构造和复制函数

a) 构造函数：初始化哈希表，将桶的个数设为100，将hasher和key\_equal的默认值传入进行构造。

b) 构造函数：声明为explicit；接受一个参数n；初始化哈希表，将桶的数量设为n，将hasher和key\_equal的默认值传入进行构造。

c) 构造函数：接受两个参数n和hf；初始化哈希表，将桶的数量设为n，哈希函数设为hf，key\_equal的默认值传入进行构造。

d) 构造函数：接受三个参数n，hf和eql；初始化哈希表，将三个参数传入进行构造。

e) 构造函数：模板函数，有一个模板参数InputIterator；接受两个参数first和last；默认的构造哈希表，调用哈希表的insert\_unique函数进行插入构造。

f) 构造函数：模板函数，有一个模板参数InputIterator；接受三个参数first，last和n；构造哈希表，调用哈希表的insert\_unique函数进行插入构造。

g) 构造函数：模板函数，有一个模板参数InputIterator；接受四个参数first，last，n和hf；构造哈希表，调用哈希表的insert\_unique函数进行插入构造。

h) 构造函数：模板函数，有一个模板参数InputIterator；接受五个参数first，last，n，hf和eql；默认的构造哈希表，调用哈希表的insert\_unique函数进行插入构造。

i) 构造函数：同样的方式只是模板参数改为了initializer\_list。

j) 复制构造函数：接受一个参数rhs；将rhs的哈希表赋给当前容器。

k) operator=：接受一个参数rhs；如果this不等于rhs，将rhs的哈希表赋给当前容器，返回this；返回值类型为unordered\_map&。

l) operator=：接受一个参数rhs(initializer\_list)；调用哈希表的clear函数，调用哈希表的reserve函数，调用哈希表的insert\_unique函数，返回this；返回值类型为unordered\_map&。

m) 析构函数：采用默认方式(default)。

2) 迭代器相关函数

a) begin函数：调用哈希表的begin函数；返回值类型为iterator。

b) begin函数：const成员函数；调用哈希表的begin函数；返回值类型为const\_iterator。

c) end函数：调用哈希表的end函数；返回值类型为iterator。

d) end函数：const成员函数；调用哈希表的end函数；返回值类型为const\_iterator。

e) cbegin函数：const成员函数；调用哈希表的cbegin函数；返回值类型为const\_iterator。

f) cend函数：const成员函数；调用哈希表的cend函数；返回值类型为const\_iterator。

3) 容量相关函数

a) empty函数：const成员函数；调用哈希表的empty函数，返回值类型为bool。

b) size函数：const成员函数；调用哈希表的size函数，返回值类型为size\_type。

c) max\_size函数：const成员函数；调用哈希表的max\_size函数；返回值类型为size\_type。

d) resize函数：接受一个参数n；调用哈希表的resize函数。

e) bucket\_count函数：const成员函数；调用哈希表的bucket\_count函数并返回；返回值类型为size\_type。

f) max\_bucket\_count函数：const成员函数；调用哈希表的max\_bucket\_count函数并返回；返回值类型为size\_type。

g) elems\_in\_bucket函数：const成员函数；接受一个参数bucket；调用哈希表的elems\_in\_bucket函数并返回；返回值类型为size\_type。

h) swap函数：接受一个参数rhs；调用哈希表的swap函数。

4) 插入删除相关函数

a) insert函数：接受一个参数value；调用哈希表的insert\_unique函数并返回，返回值类型为pair<iterator，bool>。

b) insert函数：模板函数，有一个模板参数InputIterator；接受两个参数first和last；调用哈希表的insert\_unique函数。

c) insert函数：接受两个参数first和last(const value\_type&)；调用哈希表的insert\_unique函数。

d) insert函数：接受两个参数first和last(const\_iterator)；调用哈希表的insert\_unique函数。

e) insert\_noresize函数：接受一个参数value；调用哈希表的insert\_unique函数并返回；返回值类型为pair<iterator，bool>。

f) erase函数：接受一个参数key；调用哈希表的erase函数并返回；返回值类型为size\_type。

g) erase函数：接受一个参数it；调用哈希表的erase函数。

h) erase函数：接受两个参数first和last；调用哈希表的erase函数。

i) clear函数：调用哈希表的clear函数。

5) 查找相关函数

1. find函数：接受一个参数key；调用哈希表的find函数并返回；返回值类型为iterator。
2. find函数：和上述find函数类似，const版本。
3. at函数：接受一个参数key；调用哈希表的find函数得到迭代器it，如果it的node为空，抛出std的out\_of\_range异常，否则返回it的second值；返回值类型为T&。
4. at函数：和上述at函数类似，只是换成了const版本。
5. count函数：const成员函数；接受一个参数key；调用哈希表的count函数并返回；返回值类型为size\_type。
6. equal\_range函数：接受一个参数key；调用哈希表的equal\_range函数并返回；返回值类型为pair<iterator，iterator>。
7. equal\_range函数：和上述equal\_range函数类似，只是const版本。
8. operator[]：接受一个参数key；调用哈希表的find\_or\_insert函数并返回返回值的second；返回值类型为T&。

6) 其他函数

1. operator==(inline函数)：接受两个参数lhs和rhs；判断lhs是否等于rhs。
2. operator!=(inline函数)：接受两个参数lhs和rhs；调用operator==。
3. swap函数(inline函数)：接受两个参数lhs和rhs；通过lhs调用swap函数。

### unordered\_multimap

1. unordered\_multimap

模板类，有五个模板参数Key，T，HashFcn，EqualKey和Alloc；内部包含一个哈希表ht；typedef包括allocator\_type，key\_type，data\_type，mapped\_type，value\_type，hasher，key\_equal，size\_type，difference\_type，pointer，const\_pointer，reference，const\_reference，iterator和const\_iterator；包含一个get\_allocator函数(调用哈希表的get\_allocator函数)，hash\_funct调用哈希表的hash\_funct函数并返回，key\_eq函数调用哈希表的key\_eq函数并返回；将operator==声明为友元函数。

1) 构造和复制函数

a) 构造函数：初始化哈希表，将桶的个数设为100，将hasher和key\_equal的默认值传入进行构造。

b) 构造函数：声明为explicit；接受一个参数n；初始化哈希表，将桶的数量设为n，将hasher和key\_equal的默认值传入进行构造。

c) 构造函数：接受两个参数n和hf；初始化哈希表，将桶的数量设为n，哈希函数设为hf，key\_equal的默认值传入进行构造。

d) 构造函数：接受三个参数n，hf和eql；初始化哈希表，将三个参数传入进行构造。

e) 构造函数：模板函数，有一个模板参数InputIterator；接受两个参数first和last；默认的构造哈希表，调用哈希表的insert\_equal函数进行插入构造。

f) 构造函数：模板函数，有一个模板参数InputIterator；接受三个参数first，last和n；构造哈希表，调用哈希表的insert\_equal函数进行插入构造。

g) 构造函数：模板函数，有一个模板参数InputIterator；接受四个参数first，last，n和hf；构造哈希表，调用哈希表的insert\_equal函数进行插入构造。

h) 构造函数：模板函数，有一个模板参数InputIterator；接受五个参数first，last，n，hf和eql；默认的构造哈希表，调用哈希表的insert\_equal函数进行插入构造。

i) 构造函数：同样的方式只是模板参数改为了initializer\_list。

j) 复制构造函数：接受一个参数rhs；将rhs的哈希表赋给当前容器。

k) operator=：接受一个参数rhs；如果this不等于rhs，将rhs的哈希表赋给当前容器，返回this；返回值类型为unordered\_multimap&。

l) operator=：接受一个参数rhs(initializer\_list)；调用哈希表的clear函数，调用哈希表的reserve函数，调用哈希表的insert\_equal函数，返回this；返回值类型为unordered\_multimap&。

m) 析构函数：采用默认方式(default)。

2) 迭代器相关函数

a) begin函数：调用哈希表的begin函数；返回值类型为iterator。

b) begin函数：const成员函数；调用哈希表的begin函数；返回值类型为const\_iterator。

c) end函数：调用哈希表的end函数；返回值类型为iterator。

d) end函数：const成员函数；调用哈希表的end函数；返回值类型为const\_iterator。

e) cbegin函数：const成员函数；调用哈希表的cbegin函数；返回值类型为const\_iterator。

f) cend函数：const成员函数；调用哈希表的cend函数；返回值类型为const\_iterator。

3) 容量相关函数

a) empty函数：const成员函数；调用哈希表的empty函数，返回值类型为bool。

b) size函数：const成员函数；调用哈希表的size函数，返回值类型为size\_type。

c) max\_size函数：const成员函数；调用哈希表的max\_size函数；返回值类型为size\_type。

d) resize函数：接受一个参数n；调用哈希表的resize函数。

e) bucket\_count函数：const成员函数；调用哈希表的bucket\_count函数并返回；返回值类型为size\_type。

f) max\_bucket\_count函数：const成员函数；调用哈希表的max\_bucket\_count函数并返回；返回值类型为size\_type。

g) elems\_in\_bucket函数：const成员函数；接受一个参数bucket；调用哈希表的elems\_in\_bucket函数并返回；返回值类型为size\_type。

h) swap函数：接受一个参数rhs；调用哈希表的swap函数。

4) 插入删除相关函数

a) insert函数：接受一个参数value；调用哈希表的insert\_equal函数并返回，返回值类型为iterator。

b) insert函数：模板函数，有一个模板参数InputIterator；接受两个参数first和last；调用哈希表的insert\_equal函数。

c) insert函数：接受两个参数first和last(const value\_type&)；调用哈希表的insert\_equal函数。

d) insert函数：接受两个参数first和last(const\_iterator)；调用哈希表的insert\_equal函数。

e) insert\_noresize函数：接受一个参数value；调用哈希表的insert\_equal函数并返回；返回值类型为iterator。

f) erase函数：接受一个参数key；调用哈希表的erase函数并返回；返回值类型为size\_type。

g) erase函数：接受一个参数it；调用哈希表的erase函数。

h) erase函数：接受两个参数first和last；调用哈希表的erase函数。

i) clear函数：调用哈希表的clear函数。

5) 查找相关函数

a) find函数：接受一个参数key；调用哈希表的find函数并返回；返回值类型为iterator。

b) find函数：和上述find函数类似，const版本。

c) count函数：const成员函数；接受一个参数key；调用哈希表的count函数并返回；返回值类型为size\_type。

d) equal\_range函数：接受一个参数key；调用哈希表的equal\_range函数并返回；返回值类型为pair<iterator，iterator>。

e) equal\_range函数：和上述equal\_range函数类似，只是const版本。

6) 其他函数

1. operator==(inline函数)：接受两个参数lhs和rhs；判断lhs是否等于rhs。
2. operator!=(inline函数)：接受两个参数lhs和rhs；调用operator==。
3. swap函数(inline函数)：接受两个参数lhs和rhs；通过lhs调用swap函数。
4. insert\_iterator偏特化
5. unordered\_map偏特化

模板类，有五个模板参数Key，T，HashFcn，EqualKey和Alloc；protected下维护一个容器指针；typedef包括container\_type，iterator\_category(output\_iterator\_tag)，value\_type，difference\_type，pointer和reference；构造函数接受一个参数x和两个参数x和iterator(类型)；重载operator=，接受一个参数value，调用容器的insert函数并返回自身；operator\*，operator++和operator++(int)都返回自身。

1. unordered\_multimap

和unordered\_map类似。

# 4 算法

## 4.1 数值算法

数值算法包含在文件numeric.h中。

1. accumulate函数：以初始值对每个元素做累加操作；模板函数，有两个模板参数InputIterator和T；接受三个参数first，last和init；利用for循环，对init做累加，返回init；返回值类型为T。
2. accumulate函数：以初始值对每个元素做二元操作；模板函数，有三个模板参数InputIterator，T和BinaryOperation；接受四个参数first，last，init和binary\_op；利用for循环对init做二元操作，返回init；返回值类型为T。
3. inner\_product函数：以初始值计算两个区间的内积；模板函数，有三个模板参数InputIterator1，InputIterator2和T；接受四个参数first1，last1，first2和init；利用for循环在init的基础上加上内积的值，返回init；返回值类型为T。
4. inner\_product函数：以初始值计算两个区间的内积；模板函数，有五个模板参数InputIterator1，InputIterator2，T，BinaryOperator1和BinaryOperator2；接受六个参数first1，last1，first2，init，binary\_op1和binary\_op2；用传入的二元操作利用for循环在init的基础上加上内积的值，返回init；返回值类型为T。
5. partial\_sum函数：计算局部累计求和，结构保存到以result为起始的区间上；模板函数，有两个模板参数InputIterator和OutputIterator；接受三个参数first，last和result；如果first等于last，返回result，将first的值存入result，调用\_\_partial\_sum并返回；返回值类型为OutputIterator。
6. \_\_partial\_sum函数：模板函数，有三杯模板参数InputIterator，OutputIterator和T；接受四个参数first，last，result和T\*；声明一个value等于first的值，利用while循环，当first不等于last，value加上first的值，++result存入value的值，返回++result；返回值类型为OutputIterator。
7. partial\_sum函数：计算局部累计求和，结构保存到以result为起始的区间上；模板函数，有三个模板参数InputIterator，OutputIterator和BinaryOperation；接受四个参数first，last，result和binary\_op；如果first等于last，返回result，将first的值存入result，调用\_\_partial\_sum并返回；返回值类型为OutputIterator。
8. \_\_partial\_sum函数：模板函数，有三杯模板参数InputIterator，OutputIterator和T；接受四个参数first，last，result和T\*；声明一个value等于first的值，利用while循环，当first不等于last，value的first做二元运算，++result存入value的值，返回++result；返回值类型为OutputIterator。
9. adjacent\_difference函数：计算相邻元素的差值，结构保存到以result为起始的区间上；模板函数，有两个模板参数InputIterator和OutputIterator；接受三个参数first，last和result；如果first等于last，返回result，否则调用\_\_adjacent\_difference函数并返回；返回值类型为OutputIterator。
10. \_\_adjacent\_difference函数：模板函数，有三个模板参数InputIterator，OutputIterator和T；接受四个参数first，last，result和T\*；声明一个value等于first的值，利用while循环当first不等于last时，声明一个temp等于first的值，将temp和value的差值存进result中，更新value的值，返回++result；返回值类型为OutputIterator。
11. adjacent\_difference函数：计算相邻元素的差值，结构保存到以result为起始的区间上；模板函数，有三个模板参数InputIterator，OutputIterator和BinaryOperation；接受三个参数first，last，result和binary\_op；如果first等于last，返回result，否则调用\_\_adjacent\_difference函数并返回；返回值类型为OutputIterator。
12. \_\_adjacent\_difference函数：模板函数，有四个模板参数InputIterator，OutputIterator，BinaryOperation和T；接受四个参数first，last，result，binary\_op和T\*；声明一个value等于first的值，利用while循环当first不等于last时，声明一个temp等于first的值，将temp和value进行二元运算，结果存进result中，更新value的值，返回++result；返回值类型为OutputIterator。
13. power函数(内联函数)：模幂运算；模板函数，有两个模板参数T和Integer，接受两个参数x和n；调用\_\_power函数并返回；返回值类型为T。
14. \_\_power函数：模板函数，有三个模板参数T，Integer和MonoidOperator；接受三个参数x，n和opr；如果n等于0，调用identity\_element函数并返回，利用while循环，当n的最后一位为0时，将n右移一位并将x进行二元操作；声明一个result等于x，将n右移一位，利用while循环，当n不为0时，将x进行二元操作，如果n的最后一位为1，将result和x做二元操作，将n右移一位，返回result；返回值类型为T。
15. iota函数：以value为初值递增填充first到last区间；模板函数，有两个模板参数ForwardIterator和T；接受三个参数first，last和value；利用while循环，当first不等于last时，将value的值填进first中，并将value和first递增。

## 4.2 基本算法

1. iter\_swap函数(inline函数)：将两个迭代器所指的对象交换；模板函数，有两个模板参数ForwardIterator1和ForwardIterator2；接受两个参数a和b；调用\_\_iter\_swap函数。
2. \_\_iter\_swap函数(inline函数)：模板函数，有三个模板参数ForwardIterator1，ForwardIterator和T；接受三个参数a，b和T\*；声明一个temp变量进行交换。
3. swap函数(inline函数)：交换两个变量；模板函数，有一个模板参数T，接受两个变量a和b；声明一个temp变量进行交换。
4. max函数(inline函数)：模板函数，有一个模板参数T；接受两个参数lhs和rhs；返回两者之间较大的数；返回值类型为const T&。
5. max函数(inline函数)：模板函数，有两个模板参数T和Compare；接受三个参数lhs，rhs和comp；利用comp函数进行比较，返回较大的值；返回值类型为const T&。
6. min函数(inline函数)：和max函数类似，返回较小的值。
7. min函数(inline函数)：和第二个max函数类型，返回较小的值。
8. copy函数：模板函数，有两个模板参数InputIterator和OutputIterator；接受三个参数first，last和result；调用函数copy\_aux。
9. copy\_aux函数：模板函数，有两个模板参数InputIterator和OutputIterator；接受三个参数first，last和result；调用\_\_copy函数。
10. copy\_aux函数：模板函数，有两个模板参数T和U，如果U和去掉了const类型的T是同一类型并且U有不重要的赋值复制函数，函数的返回类型为U\*(用到了enable\_if，is\_same，remove\_const和is\_trivially\_copy\_assignable)；接受三个参数first，last和result；算出last和first之间的距离n，如果n不为0，调用memmove函数进行复制，返回result+n；返回值类型为U\*。
11. \_\_copy函数(input\_iterator版本)：模板函数，有两个模板参数InputIterator和OutputIterator；接受四个参数first，last，result和input\_iterator\_tag；利用for循环将first的值赋给result，返回result；返回值类型为OutputIterator。
12. \_\_copy函数(random\_access\_iterator版本)：模板函数，有两个模板参数RandomAccessIterator和OutputIterator；接受四个参数first，last，result和random\_access\_iterator\_tag；利用for循环(n来判断)，将first的值赋给result，返回result；返回值类型为OutputIterator。
13. copy\_backward函数：模板函数，有两个模板参数BidirectionalIterator1和BidirectionalIterator2；接受三个参数first，last和result；调用copy\_backward\_aux函数。
14. copy\_backward\_aux函数：模板函数，有两个模板参数BidirectionalIterator1和BidirectionalIterator2；接受三个参数first，last和result；调用\_\_copy\_backward函数。
15. copy\_backward\_aux函数(trivially\_copy\_assignable特供版本)；模板函数，有两个模板参数T和U；接受三个参数first，last和result；算出first和last之间的距离n，如果n不为0，将result减n，然后调用memmove函数，返回result；返回值类型用enable\_if决定。
16. \_\_copy\_backward函数(bidirectional\_iterator版本)：模板函数，有两个模板参数BidirectionalIterator1和BidirectionalIterator2；接受四个参数first，last，result和bidirectional\_iterator\_tag；利用while循环，将last的值赋给result，返回result；返回值类型为BidirectionalIterator2。
17. \_\_copy\_backward函数(random\_access\_iterator版本)：模板函数，有两个模板参数RandomAccessIterator1和BidirectionalIterator2；接受四个参数first，last，result和bidirectional\_iterator\_tag；利用while循环(判断n)，将last的值赋给result，返回result；返回值类型为BidirectionalIterator2。
18. copy\_if函数：模板函数，有三个模板参数InputIterator，OutputIterator和UnaryPredicate；接受四个参数first，last，result和unary\_pred；利用for循环，如果first的值满足unary\_pred，就将first的值赋给result，返回result；返回值类型为OutputIterator。
19. copy\_n函数：模板函数，有三个模板参数InputIterator，Size和OutputIterator；接受三个参数first，count和result；调用\_\_copy\_n函数并返回；返回值类型为pair<InputIterator，OutputIterator>。
20. \_\_copy\_n函数(input\_iterator版本)：模板函数，有三个模板参数InputIterator，Size和OutputIterator；接受四个参数first，count，result和input\_iterator\_tag；利用for循环将first的值赋给result；返回值类型为pair<InputIterator，OutputIterator>。
21. \_\_copy\_n函数(random\_access\_iterator版本)：模板函数，有三个模板参数RandomAccessIterator，Size和OutputIterator；接受四个参数first，count，result和random\_access\_iterator\_tag；调用copy函数并返回；返回值类型为pair<RandomAccessIterator，OutputIterator>。
22. fill函数：模板函数，有两个模板参数ForwardIterator和T；接受三个参数first，last和value；调用\_\_fill函数。
23. \_\_fill函数(forward\_iterator版本)：模板函数，有两个模板参数ForwardIterator和T；接受四个参数first，last，value和forward\_iterator\_tag；利用for循环，将value赋给first。
24. \_\_fill函数(random\_access\_iterator版本)：模板函数，有两个模板参数RandomAccessIterator和T；接受四个参数first，last，value和random\_access\_iterator；调用fill\_n函数。
25. fill\_n函数：模板函数，有三个模板参数OutputIterator，Size和T；接受三个参数first，n和value；调用\_\_fill\_n函数并返回；返回值类型为OutputIterator。
26. \_\_fill\_n函数：模板函数，有三个模板参数OutputIterator，Size和T；接受三个参数first，n和value；利用for循环，将value值赋给first，返回first；返回值类型为OutputIterator。
27. \_\_fill\_n函数(one\_type类型特化版本)：模板函数，有三个模板参数T，Size和U；接受三个参数first，n和value；如果n大于0，调用memset函数(value转为unsigned char，n转为size\_t)，返回first+n；返回值类型为T\*；存在的条件为T为整数类型(is\_integral)，T的大小为一字节，T不是bool类型(is\_same)，U是整数类型并且U的大小为一字节。
28. mismatch函数：平行比较两个序列，找到第一处失配的元素；模板函数，有两个模板参数InputIterator1和InputIterator2；接受三个参数first1，last1和first2；利用while循环找到第一个不相等的元素，将两个first1和first2打包为pair返回；返回值类型为pair<InputIterator1，InputIterator2>。
29. mismatch函数：模板函数，有三个模板参数InputIterator1，InputIterator2和BinaryPredicate；接受四个参数first1，last1，first2和binary\_pred；利用while循环找到第一个不符合条件的元素，将两个first1和first2打包为pair返回；返回值类型为pair<InputIterator1，InputIterator2>。
30. equal函数：模板参数，有两个模板参数InputIterator1和InputIterator2；接受三个参数first1，last1和first2；利用for循环进行比较，如果不相等返回false，否则返回true；返回值类型为bool。
31. equal函数：模板参数，有三个模板参数InputIterator1，InputIterator2和BinaryPredicate；接受四个参数first1，last1，first2和binary\_pred；利用for循环进行比较，如果不符合条件返回false，否则返回true；返回值类型为bool。
32. lexicographical\_compare函数：模板函数，有两个模板参数InputIterator1和InputIterator2；接受四个参数first1，last1，first2和last2；利用for循环遍历，如果first1的值小于first2的值，返回true；如果first1的值大于first2的值，返回false；如果到达 last1 而尚未到达 last2 返回 true；如果到达 last2 而尚未到达 last1 返回 false(用一个表达式完成)；返回值类型为bool。
33. lexicographical\_compare函数：模板函数，有三个模板参数InputIterator1，InputIterator2和Compare；接受五个参数first1，last1，first2，last2和comp；利用for循环遍历，如果first1的值和first2的值符合条件，返回true；如果first2的值和first2的值符合条件，返回false；如果到达 last1 而尚未到达 last2 返回 true；如果到达 last2 而尚未到达 last1 返回 false(用一个表达式完成)；返回值类型为bool。
34. lexicographical\_compare函数(const unsigned char\*特化)：接受四个参数first1，last1，first2和last2；算出两个序列的长度分别为len1和len2，先利用memcpy函数比较相同长度的部分，如果相等(result为0)长度较大的比较大，不相等返回result<0的值；返回值类型为bool。
35. lexicographical\_compare\_3way函数：模板函数，有两个模板参数InputIterator1和InputIterator2；接受四个参数first1，last1，first2和last2；利用for循环遍历，如果first1的值小于first2的值，返回-1；如果first1的值大于first2的值，返回1；如果first2等于last2返回!(first1==last1)，否则返回-1；返回值类型为int。
36. lexicographical\_compare\_3way函数(const unsigned char\*特化版本)：模板函数，有两个模板参数InputIterator1和InputIterator2；接受四个参数first1，last1，first2和last2；算出两个序列的长度分别为len1和len2，先利用memcpy函数比较相同长度的部分，如果result不为0返回result，如果len1等于len2返回0，若len1小于len2返回-1，否则返回1；返回值类型为int。

## set相关算法

1. set\_union函数：模板函数，有三个模板参数InputIterator1，InputIterator2和OutputIterator；接受五个参数first1，last1，first2，last2和result；利用while循环遍历两个set中的元素，如果first1的值小于first2的值，将first1的值赋给result并做相应调整，如果first2的值小于first1的值，将first2的值赋给result并做相应调整，如果first1的值等于first2的值，将first1的值赋给result并做相应调整，再调用copy函数将剩余的元素拷贝到result上并返回；返回值类型为OutputIterator。
2. set\_union函数：模板函数，有四个模板参数InputIterator1，InputIterator2，OutputIterator和Compare；接受六个参数first1，last1，first2，last2，result和comp；和前面的set\_union函数类似，只是比较换成了comp函数；返回值类型为OutputIterator。
3. set\_intersection函数：模板函数，有三个模板参数InputIterator1，InputIterator2和OutputIterator；接受五个参数first1，last1，first2，last2和result；利用while循环遍历两个set中的元素，如果first1的值等于first2的值，将first1的值赋给result并做相应调整，返回result；返回值类型为OutputIterator。
4. set\_intersection函数：模板函数，有四个模板参数InputIterator1，InputIterator2，OutputIterator和Compare；接受六个参数first1，last1，first2，last2，result和comp；和前面的set\_intersection函数类似，只是比较换成了comp函数；返回值类型为OutputIterator。
5. set\_difference函数：模板函数，有三个模板参数InputIterator1，InputIterator2和OutputIterator；接受五个参数first1，last1，first2，last2和result；利用while循环遍历两个set中的元素，将first1中有而first2中没有的元素拷贝到result中；返回值类型为OutputIterator。
6. set\_difference函数：模板函数，有四个模板参数InputIterator1，InputIterator2，OutputIterator和Compare；接受六个参数first1，last1，first2，last2，result和comp；和前面的set\_difference函数类似，只是比较换成了comp函数；返回值类型为OutputIterator。
7. set\_symmetric\_difference函数：模板函数，有三个模板参数InputIterator1，InputIterator2和OutputIterator；接受五个参数first1，last1，first2，last2和result；利用while循环遍历两个set中的元素，将两个set中不同的元素拷贝到result中；返回值类型为OutputIterator。
8. set\_symmetric\_difference函数：模板函数，有四个模板参数InputIterator1，InputIterator2，OutputIterator和Compare；接受六个参数first1，last1，first2，last2，result和comp；和前面的set\_symmetric\_difference函数类似，只是比较换成了comp函数；返回值类型为OutputIterator。

## 4.4 heap相关算法

1. push\_heap
2. push\_heap：模板函数，模板参数为RandomAccessIterator，接受两个参数first和last(表示heap底层容器的头尾，并且新元素已经插入到底层容器的最尾端)；实现上调用辅助函数push\_heap\_aux。
3. push\_heap\_aux：模板参数，有三个模板参数为RandomAccessIterator，Distance和T，接受四个参数first，last，Distance\*和T\*；实现上调用辅助函数\_\_push\_heap，传入的参数分别是first，新插入元素编号，头部元素编号和新插入元素的值。
4. \_\_push\_heap：模板函数，有三个模板参数为RandomAccessIterator，Distance和T，接受四个参数first，holeIndex，topIndex和value；首先计算出父节点的编号，利用while循环当未到达顶端并且父节点的值小于新插入元素的值，将洞值赋给父节点，调整洞号，计算新的父节点编号；最后将value赋给洞号结点。
5. 自定义比较的push\_heap：在上述三个函数的基础上多了一个新的参数comp，在最后比较父节点和插入元素大小的使用comp比较。
6. pop\_heap(pop之后最大的元素只是放在底层容器的最尾端)
7. pop\_heap：模板函数，模板参数为RandomAccessIterator，接受两个参数first和last；实现上调用辅助函数pop\_heap\_aux。
8. pop\_heap\_aux：模板函数，模板参数为RandomAccessIterator和T，接受三个参数first，last和T\*；实现上调用辅助函数\_\_pop\_heap。
9. \_\_pop\_heap函数：模板函数，模板参数为RandomAccessIterator，T和Distance，接受五个参数first，last，result，value和Distance\*；实现上首先将result位置赋值为first的值，然后调用辅助函数adjust\_heap。
10. adjust\_heap：模板函数，模板参数为RandomAccessIterator，T和Distance，接受四个参数first，holeIndex，len和value；首先算出洞结点右子结点的编号，利用while循环(如果洞结点左右子结点都有)，比较左右子结点大小，得到值更大的那一个子结点编号，然后将该值赋给洞结点，更新洞结点和右子结点的编号；如果洞结点只有左子结点，将左子结点的值赋给洞结点并更新洞结点编号；最后将value赋给洞结点。
11. 自定义比较的pop\_heap：在上述四个函数的基础上多了一个新的参数comp，在最后比较父节点和插入元素大小的使用comp比较。
12. sort\_heap(排序之后原来的heap就不是一个合法的heap)
13. sort\_heap：模板函数，模板参数为RandomAccessIterator，接受两个参数first和last；利用for循环，当heap中还多余1个元素时，调用pop\_heap函数并将last--。
14. 自定义比较的sort\_heap：在上述函数的基础上多了一个新的参数comp，调用自定义的pop\_heap函数。
15. make\_heap
16. make\_heap：模板函数，模板参数为RandomAccessIterator，接受两个参数first和last；实现上调用辅助函数\_\_make\_heap。
17. \_\_make\_heap：模板函数，模板参数为RandomAccessIterator，T和Distance，接受四个参数first，last，T\*和Distance\*；计算出元素个数，如果小于2则直接返回；算出第一个需要下调的结点编号(len - 2) / 2，利用while循环不断调用\_\_adjust\_heap函数并调整holeindex的值。
18. 自定义比较的make\_heap：在上述两个函数的基础上多了一个新的参数comp，调用自定义的\_\_adjust\_heap函数。

## 其它算法

1. all\_of函数：检查[first, last)内是否全部元素都满足一元操作unary\_pred为 true的情况；模板函数，有两个模板参数InputIterator和UnaryPredicate；接受三个参数first，last和unary\_pred；如果for循环遍历，如果\*first的值不符合unary\_pred返回false，否则返回true；返回值类型为bool。
2. any\_of函数：检查[first，last)内是否存在元素满足一元操作unary\_pred为true的情况；模板函数，有两个模板参数InputIterator和UnaryPredicate；接受三个参数first，last和unary\_pred；如果for循环遍历，如果first的值符合unary\_pred返回true，否则返回false；返回值类型为bool。
3. none\_of函数：检查[first，last)内是否全部元素都不满足一元操作unary\_pred为true的情况；模板函数，有两个模板参数InputIterator和UnaryPredicate；接受三个参数first，last和unary\_pred；如果for循环遍历，如果first的值符合unary\_pred返回false，否则返回true；返回值类型为bool。
4. count函数：模板函数，有两个模板参数InputIterator和T；接受三个参数first，last和value；利用for循环遍历，如果first的值等于value，++n，返回n；返回值类型为InputIterator的difference\_type。
5. count\_if函数：模板函数，有三个模板参数InputIterator，T和UnaryPredicate；接受三个参数first，last，和unary\_pred；利用for循环遍历，如果first的值满足unary\_pred的条件，++n，返回n；返回值类型为InputIterator的difference\_type。
6. find函数：模板函数，有两个模板参数InputIterator和T；接受三个参数first，last和value；调用\_\_find函数并返回；返回值类型为InputIterator。
7. \_\_find函数(input iterator版本)：模板函数，有两个模板参数InputIterator和T；接受四个参数first，last，value和input\_iterator\_tag；利用while循环，找到第一个等于value的迭代器并返回；返回值类型为InputIterator。
8. \_\_find函数(random\_iterator版本)：利用循环展开优化算法；模板函数，有两个模板参数RandomAccessIterator和T；接受四个参数first，last，value和input\_iterator\_tag；首先得到first和last之间的距离n(注意类型)，将n左移2位(分为4组)，利用for循环依次查找每组中是否存在等于value的元素，如果存在返回first，最后利用switch判断剩下的一组中是否存在等于value的元素，如果不存在返回last；返回值类型为RandomAccessIterator。
9. find\_if函数：模版函数，有两个模板参数InputIterator和UnaryPredicate；接受三个参数first，last和unary\_pred；调用\_\_find\_if函数并返回；返回值类型为InputIterator。
10. \_\_find\_if函数(input iterator版本)：和input iterator版本的\_\_find函数类似，只是判断条件变为了unary\_pred。
11. \_\_find\_if函数(random iterator版本)：和random iteraor版本的\_\_find函数类似，只是判断条件变为了unary\_pred。
12. find\_if\_not函数：模版函数，有两个模板参数InputIterator和UnaryPredicate；接受三个参数first，last和unary\_pred；调用\_\_find\_if\_not函数并返回；返回值类型为InputIterator。
13. \_\_find\_if\_not函数(input iterator版本)：和input iterator版本的\_\_find函数类似。
14. \_\_find\_if\_not函数(random iterator版本)：和random iterator版本的\_\_find函数类似。
15. search函数：区间搜索；模板函数，接受两个模板参数ForwardIterator1和ForwardIterator2；接受四个参数first1，last1，first2和last2；如果两个区间有一个为空区间，返回first1；如果第二个区间只有一个元素，调用find函数并返回；声明一个p1为first2的下一个元素，current为first1，利用while循环，当first1不等于last1时，调用find函数寻找first2的值并将结果赋给first1，如果first1等于last1，返回last1；将first1赋给current，如果current的下一个元素为last1，返回last1；利用while循环，当current的值等于p的值时，如果p的下一个元素等于last2，返回last1；如果current的下一个元素等于last1，返回last1；++first1；返回first1；返回值类型为FowardIterator1。
16. search函数：模板函数，接受三个模板参数ForwardIterator1，ForwardIterator2和BinaryPredicate；接受五个参数first1，last1，first2，last2和binary\_pred；如果两个区间有一个是空区间，返回last1；如果第二个区间只有一个元素，遍历第一个区间进行寻找；声明一个迭代器p等于first2的下一个元素，current等于first1，利用while循环，当first1不等于last1时，利用while循环找到第一个first1和first2相等的元素，如果first1等于last1，返回last1；将first1赋给current，如果current的下一个元素为last1，返回last1；利用while循环，当current的p符合binary\_pred时，如果p的下一个元素为last2，返回last1，如果current的下一个元素为last1，返回last1，++first1，返回first1；返回值类型为ForwardIterator1。
17. search\_n函数：在[first, last)中查找连续 n 个 value 所形成的子序列；模板函数，有三个模板参数ForwardIterator，Integer和T；接受四个参数first，last，count和value；如果count小于等于0，返回first；调用find函数查找第一个等于value的元素并将返回值赋给first，利用while循环，当first不等于last时，查找是否连续存在元素，如果存在返回first，否则调用find函数寻找下一个等于value的元素，没有找到返回last；返回值类型为ForwardIterator。
18. search\_n函数：模板函数，有四个模板参数ForwardIterator，Integer，T和BinaryPredicate；接受五个参数first，last，count，value和binary\_pred；实现和前面的search\_n函数类似，只是把调用find函数换成了用while循环查找。
19. find\_end函数：在[first1, last1)区间中查找[first2, last2)最后一次出现的地方；模板函数，有两个模板参数ForwardIterator1和ForwardIterator2；接受四个参数first1，last1，first2和last2；调用\_\_find\_end函数并返回；返回值类型为ForwardIterator1。
20. \_\_find\_end函数(forward iterator版本)：模板函数，有两个模板参数ForwardIterator1和ForwardIterator2；接受六个参数first1，last1，first2，last2，forward\_iterator\_tag和forward\_iterator\_tag；如果第二个区间为空，返回last1；声明一个迭代器result等于last1，利用while true循环，调用search函数并将返回值赋给new\_result，如果new\_result等于result，返回result，否则将new\_result赋给result，first1为result的下一个元素；返回值类型为ForwardIterator1。
21. \_\_find\_end函数(bidirectional iterator版本)：模板函数，有两个模板参数BidirectionalIterator1和BidirectionalIterator2；接受六个参数first1，last1，first2，last2，bidirectional\_iterator\_tag和bidirecitional\_iterator\_tag；利用first1和first2声明两个返回迭代器rlast1和rlast2，调用search函数并将返回值赋给rresult，如果rresult等于rlast1，返回last1，否则获得rresult的正向迭代器result(base函数)；利用advance函数将result向前移动第二个区间的长度，返回result；返回值类型为BidirectionalIterator1。
22. find\_end函数：模板函数，有三个模板参数ForwardIterator1，ForwardIterator2和BinaryPredicate；接受五个参数first1，last1，first2，last2和binary\_pred；调用\_\_find\_end函数。
23. \_\_find\_end函数(forward iterator版本)：模板函数，有三个模板参数ForwardIterator1，ForwardIterator2和BinaryPredicate；接受七个参数first1，last1，first2，last2，binary\_pred，forward\_iterator\_tag和forward\_iterator\_tag；和前面的\_\_find\_end函数类似。
24. \_\_find\_end函数(bidirectional iterator版本)：模板函数，有三个模板参数BidirectionalIterator1，BidirectionalIterator2和BinaryPredicate；接受七个参数first1，last1，first2，last2，binary\_pred，bidirectional\_iterator\_tag和bidirecitional\_iterator\_tag；和前面的\_\_find\_end函数类似。
25. find\_first\_of函数：在[first1, last1)中查找[first2, last2)中的某些元素；模板函数，有两个模板参数InputIterator和ForwardIterator；接受四个参数first1，last1，first2和last2；利用两个for循环进行查找，找到返回first1，没找到返回last1；返回值类型为InputIterator。
26. find\_first\_of函数：模板函数，有三个模板参数InputIterator，ForwardIterator和BinaryPredicate；接受五个参数first1，last1，first2，last2和binary\_pred；利用两个for循环进行查找，判断条件为binary\_pred，找到返回first1，没找到返回last1；返回值类型为InputIterator。
27. for\_each函数：使用一个函数对象 f 对[first, last)区间内的每个元素执行一个 operator() 操作，但不能改变元素内容；模板函数，有两个模板参数InputIterator和Function；接受三个参数first，last和f；利用for循环对每一个first值调用f函数，返回f；返回值类型为Function。
28. adjacent\_find函数：找出第一对匹配的相邻元素；模板函数，有一个模板参数ForwardIterator；接受两个参数first和last；如果区间为空，返回last，声明一个next等于first，利用while循环，当++next不等于last时，如果first的值等于next的值，返回first，否则first等于next，没有匹配的返回last；返回值类型为ForwardIterator。
29. adjacent\_find函数：模板函数，有两个模板参数ForwardIterator和BinaryPredicate；接受三个参数first，last和binary\_pred；和上面的adjacent\_find函数类似，只是比较条件换成了binary\_pred；返回值类型为ForwardIterator。
30. lower\_bound函数：模板函数，有两个模板参数FowwardIterator和T；接受三个参数first，last和value；调用函数\_\_lower\_bound并返回，返回值类型为ForwardIterator。
31. \_\_lower\_bound(forward iterator版本)函数：模板函数，有两个模板参数ForwardIterator和T；接受四个参数first，last，value和forward\_iterator\_tag；首先调用distance函数算出first和last之间的距离len，声明变量half和middle，利用while循环，当len大于0时，half等于len的一半，middle为first移动half的距离(distance函数)，如果middle的值小于value，让first等于middle，并让first++，len等于len-half-1，否则len等于half，返回first；返回值类型为ForwardIterator。
32. \_\_lower\_bound(random iterator版本)函数：模板函数，有两个模板参数RandomAccessIterator和T；接受四个参数first，last，value和random\_iterator\_tag；和forward iterator版本类似，只是距离移动直接进行。
33. lower\_bound函数：和前面的lower\_bound类似，只是多了一个比较函数comp。
    * 1. \_\_lower\_bound函数(forward iterator版本)：和前面forward iterator版本的\_\_lower\_bound函数类似，只是多了一个比较函数comp。
34. \_\_lower\_bound函数(random iterator版本)：和前面的random iterator版本的\_\_lower\_bound函数类似，只是多了一个比较函数comp。
35. upper\_bound函数：模板函数，有两个模板参数FowwardIterator和T；接受三个参数first，last和value；调用函数\_\_upper\_bound并返回，返回值类型为ForwardIterator。
36. \_\_upper\_bound(forward iterator版本)函数：模板函数，有两个模板参数ForwardIterator和T；接受四个参数first，last，value和forward\_iterator\_tag；首先调用distance函数算出first和last之间的距离len，声明变量half和middle，利用while循环，当len大于0时，half等于len的一半，middle为first移动half的距离(distance函数)，如果value小于middle的值，让len等于half，否则让first等于middle，并让first++，len等于len-half-1，返回first；返回值类型为ForwardIterator。
37. \_\_upper\_bound(random iterator版本)函数：模板函数，有两个模板参数RandomAccessIterator和T；接受四个参数first，last，value和random\_iterator\_tag；和forward iterator版本类似，只是距离移动直接进行。
38. upper\_bound函数：和前面的upper\_bound类似，只是多了一个比较函数comp。
39. \_\_upper\_bound函数(forward iterator版本)：和前面forward iterator版本的\_\_upper\_bound函数类似，只是多了一个比较函数comp。
40. \_\_upper\_bound函数(random iterator版本)：和前面的random iterator版本的\_\_upper\_bound函数类似，只是多了一个比较函数comp。
41. binary\_search函数：模板函数，有两个模板参数FowardIterator和T；接受三个参数first，last和value；调用lower\_bound函数得到返回值i，判断i是否等于last并且value是否等于i的值；返回值类型为bool。
42. binary\_search函数：和前面的binary\_search函数类似，只是多了comp函数。
43. equal\_range函数：模板函数，有两个模板参数ForwardIterator和T；接受三个参数first，last和value；调用\_\_euqal\_range函数并返回；返回值类型为pair<ForwardIterator，ForwardIterator>。
44. \_\_equal\_range(forward iterator版本)：模板函数，有两个模板参数ForwardIterator和T；接受四个参数first，last，value和forward\_iterator\_tag；首先调用distance函数算出first和last之间的距离len，声明变量half，middle，left和right，利用while循环，当len大于0时，half为len的一半，middle为first移动half的距离，如果middle的值小于value，first等于middle一下个元素，len等于len-half-1，如果value小于middle的值，len等于half，否则调用lower\_bound函数得到第一个不小于value的元素位置left(区间为first到middle)，将first移动到当前区间末尾(first+len)，调用upper\_bound函数得到第一个大于value的元素位置right(区间为++middle到first)，返回left和right，如果没有找到返回两个last；返回值类型为pair<ForwardIterator，ForwardIterator>。
45. \_\_equal\_range(random iterator版本)：和forward iterator版本的\_\_equal\_range函数类似，只是元素移动上不同。
46. equal\_range函数：和前面的equal\_range函数类似，只是多了一个比较函数comp。
47. \_\_equal\_range函数(forward iterator版本)：和前面forward iterator版本的\_\_equal\_range函数类似，只是多了一个比较函数comp。
48. \_\_equal\_range函数(random iterator版本)：和前面random iterator版本的\_\_equal\_range函数类似，只是多了一个比较函数comp。
49. generate函数：模板函数，有两个模板参数ForwardIterator和Generator；接受三个参数first，last和gen；利用for循环对first调用gen函数。
50. generate\_n函数：模板函数，有三个模板参数ForwardIterator，Size和Generator；接受三个参数first，n和gen；利用for循环对first调用gen函数。
51. includes函数：模板函数，有两个模板参数InputIterator1和InputIterator2；接受四个参数first1，last1，first2和last2；利用while循环，当两个序列没有结束时，如果first2的值小于first1的值，返回false，如果first1的值小于first2的值，++first2，否则++first1，++first2，最后判断序列2是否遍历结束并返回；返回值类型为bool。
52. includes函数：和前面的includes函数类似，只是多了一个比较函数comp。
53. is\_heap函数：模板函数，有一个模板参数RandomAccessIterator；接受两个参数first和last；先调用distance函数算出first和last之间的距离n，声明一个变量parent为0，利用for循环，当child不为n时，如果parent的值小于child的值，返回false，如果两个孩子都判断结束(child是否为偶数)，++parent，遍历结束返回true；返回值类型为bool。
54. is\_heap函数：和前面的is\_heap函数类似，只是多了一个比较函数comp。
55. is\_sorted函数：模板函数，有一个模板参数ForwardIterator；接受两个参数first和last；如果序列为空，返回true，声明一个next为first的下一个元素，利用for循环，当next不等于last时，如果next的值小于first的值，返回false，遍历结束返回true；返回值类型为bool。
56. is\_sorted函数：和前面的is\_sorted函数类似，只是多了一个比较函数comp。
57. median函数：模板函数，有一个模板参数T；接受三个参数left，mid和right；分成6种情况用if-else判断；返回值类型为bool。
58. median函数：和前面的median函数类似，只是多了一个比较函数comp。
59. max\_element函数：模板函数，有一个模板参数ForwardIterator；接受两个参数first和last；如果序列为空，返回first，利用while循环依次遍历并记录最大值result并返回；返回值类型为ForwardIterator。
60. max\_element函数：和前面的max\_element函数类似，只是多了一个比较函数comp。
61. min\_element函数：模板函数，有一个模板参数ForwardIterator；接受两个参数first和last；如果序列为空，返回first，利用while循环依次遍历并记录最小值result并返回；返回值类型为ForwardIterator。
62. min\_element函数：和前面的min\_element函数类似，只是多了一个比较函数comp。
63. swap\_ranges函数：模板函数，有两个模板参数ForwardIterator1和ForwardIterator2；接受三个参数first1，last1和first2；利用for循环，当序列没有遍历结束时，调用iter\_swap函数进行交换，遍历结束返回first2；返回值类型为ForwardIterator2。
64. transform函数：模板函数，有三个模板参数InputIterator，OutputIterator和UnaryOperation；接受四个参数first，last，result和unary\_op；利用for循环进行遍历，将first的值经过unary\_op操作后存入result中，返回result；返回值类型为OutputIterator。
65. transform函数：和前面的transform函数类似，只是操作变为了二元操作binary\_op(类型为BinaryOperation)，多了一个区间起点first2(类型为InputIterator2)。
66. remove\_copy函数：模板函数，有三个模板参数InputIterator，OutputIterator和T；接受四个参数first，last，result和value；利用for循环遍历整个序列，如果first的值不等于value，将first的值拷贝到result中，返回result；返回值类型为OutputIterator。
67. remove函数(并不真正将元素从容器删除)：模板函数，有两个模板参数ForwardIterator和T；接受三个参数first，last和value；调用find函数找到第一个等于value的元素位置first，声明一个next为first，如果first等于last返回last，否则调用remove\_copy函数(从next的下一个元素到last)并返回；返回值类型为ForwardIterator。
68. remove\_copy\_if函数：和remove\_copy函数类似，只是多了一个判断函数unary\_pred(类型为UnaryPredicate)。
69. remove\_if函数：和remove函数类似，只是多了一个判断函数unary\_pred(类型为UnaryPredicate)。
70. replace函数：模板函数，有两个模板参数ForwardIterator和T；接受四个参数first，last，old\_value和new\_value；利用for循环遍历整个序列，将等于old\_value的元素换成new\_value。
71. replace\_copy函数：模板函数，有三个模板参数InputIterator，OutputIterator和T；接受五个参数first，last，result，old\_value和new\_value；利用for循环遍历整个序列，将调整的序列拷贝到result中，返回result；返回值类型为OutputIterator。
72. replace\_if函数：和replace函数类型，只是多了一个判断函数unary\_pred(累心为UnaryPredicate)。
73. replace\_copy\_if函数：和replace\_copy函数类似，只是多了一个判断函数unary\_pred(类型为UnaryPredicate)。
74. reverse函数：模板函数，有一个模板参数BidirectionalIterator；接受两个参数first和last；调用\_\_reverse函数。
75. \_\_reverse函数(bidirecitional iterator版本)：模板函数，有一个模板参数BidirectionalIterator；接受三个参数first，last和bidirectional\_iterator\_tag；利用一个while true循环，当左右指针所指元素相同或只相差一个位置时return，调用iter\_swap函数进行交换。
76. \_\_reverse函数(random iterator版本)：模板函数，有一个模板参数RandomAccessIterator；接受三个参数first，last和random\_access\_iterator\_tag；利用while循环，当first小于last时，利用iter\_swap函数进行交换。
77. reverse\_copy函数：模板函数，有两个模板参数BidirectionalIterator和OutputIterator；接受三个参数first，last和result；利用while循环，当first不等于last时，从last当first将序列的值拷贝到result中，遍历结束返回result；返回值类型为OutputIterator。
78. random\_shuffle函数：模板函数，有一个模板参数RandomAccessIterator；接受两个参数first和last；如果序列为空直接返回，调用srand设定随机数种子，利用for循环，从第二个元素开始，将当前元素和之前的一个随机元素调用iter\_swap函数进行互换。
79. random\_shuffle函数：模板函数，有两个模板参数RandomAccessIterator和RandomNumberGenerator；接受三个参数first，last和rand；如果序列为空直接返回，调用distance函数算出first和last之间的距离len，利用for循环，从第二个元素开始，将当前元素和之前的一个随机元素(rand(i – first + 1) % len)调用iter\_swap函数进行互换。
80. rotate函数：模板函数，有一个模板参数ForwardIterator；接受三个参数first，middle和last；如果middle等于first，返回last，如果middle等于last，返回first，调用函数\_\_rotate并返回；返回值类型为ForwardIterator。
81. \_\_rotate函数(forward iterator版本)：模板函数，有一个模板参数ForwardIterator；接受四个参数first，middle，last和forward\_iterator\_tag；middle将序列分为两个子序列，声明一个变量first2为middle(第二个序列的开头)，利用do-while循环将后半部分序列移动到前面，移动过程中如果前半部分序列处理结束(first等于middle)，将middle设为first2重新划分序列，循环结束后将first的值设为新变量new\_middle(函数的返回值)，开始处理剩余部分，将first2设为middle，利用while循环，当first2不等于last时，交换两半部分的元素，过程中如果前半部分先结束(first等于middle)，将middle设为first2继续划分，如果后半部分先结束(first2等于last)，将first2设为middle继续划分；返回值类型为ForwardIterator。
82. \_\_rotate函数(bidirectional iterator版本)：模板函数，有一个模板参数BidirecitionalIterator；接受四个参数first，middle，last和bidirectional\_iterator\_tag；总体思想是进行三次翻转；首先调用reverse函数将两个区间进行翻转，利用while循环对收尾元素进行交换，当有一个区间结束时，调用reverse函数对另一个区间剩下的元素进行翻转并返回；返回值类型为BidirectionalIterator。
83. \_\_gcd函数：模板函数，有一个模板参数EuclideanRingElement；接受两个参数m和n；利用辗转相除法求最大公因数；利用while循环，当n不为0时，得到t为m%n，令m为n，n为t，循环结束返回m；返回值类型为EuclideanRingElement。
84. \_\_rotate函数(random access iterator版本)：模板函数，有一个模板参数RandomAccessIterator；接受四个参数first，middle，last和random\_access\_iterator\_tag；首先算出整个序列的长度n和前后两段的长度l和r，算出返回的位置result(first + last - middle)，如果l等于r，调用swap\_ranges函数进行交换并返回，调用\_\_gcd函数算出需要大循环的次数cycle\_times(因为以步长l遍历长度为n的序列需要gcd(l，n)次才能全部遍历)，利用for进行大循环，声明value为first的值，p为first，如果l小于r，利用for进行r/cycle\_times次循环(一次小循环需要把后半部分中r/cycle\_times个元素移动到前半部分)，每次遍历中，如果p大于first+r(需要重复遍历序列)，重头遍历序列并调整p，依次进行步长为l的遍历，如果l大于等于r，利用for进行l/cycle\_times-1次循环(因为最后一步赋值相当于把前半部分元素移动到后半部分，所以需要-1)，在循环中，如果p满足条件，将后半部分元素移动到前半部分，每次循环将前半部分元素移动到后半部分，小循环结束后将value的值赋给最后位置，++first，大循环结束后返回result；返回值类型为RandomAccessIterator。
85. rotate\_copy函数：模板函数，有两个模板参数ForwardIterator和OutputIterator；接受四个参数first，middle，last和result；利用两个copy函数分别将后半部分和前半部分元素移动到指定位置上；返回值类型为OutputIterator。
86. is\_permutation函数：模板函数，有两个模板参数ForwardIterator1和ForwardIterator2；接受四个参数first1，last1，first2和last2；将两个模板的value\_type定义为v1和v2，利用static\_assert保证两个类型相同，调用\_\_is\_permutation函数并返回；返回值类型为bool。
87. is\_permutation函数：模板函数，有三个模板参数ForwardIterator1，ForwardIterator2和BinaryPred；接受五个参数first1，last1，first2，last2和pred；调用\_\_is\_permutation函数并返回；返回值类型为bool。
88. \_\_is\_permutation函数：模板函数，有三个模板参数ForwardIterator1，ForwardIterator2和BinaryPred；接受五个参数first1，last1，first2，last2和pred；利用for循环找到两个序列相同的前缀，调用distance函数算出两个序列剩余的长度len1和len2，如果长度都为0返回true，如果长度不相等返回false，利用for循环遍历第一个序列剩余的元素，判断当前遍历的元素是否在之前的序列中出现过，如果没有出现过(之前没有判断过这个元素)，算出当前元素在第二个序列中出现的次数c2，如果c2为0，返回false，算出当前元素在第一个序列中出现的次数c1，如果c1不等于c2，返回false，循环结束后返回true；返回值类型为bool。
89. next\_permutation函数：模板函数，有一个模板参数BidirectionalIterator；接受两个参数first和last；如果序列为空或只有一个元素，返回false，声明一个迭代器i指向最后一个元素，利用while true循环，声明一个迭代器ii等于i，--i，如果i的值比ii的值小，声明一个迭代器j等于last，利用while循环找到第一个大于i的值的j，调用iter\_swap函数交换i和j，调用reverse函数将ii到last序列进行逆转，返回true，如果i等于first，调用reverse函数逆转整个序列并返回false；返回值类型为bool。
90. next\_permutation函数：和前一个next\_permutation函数类似，只是多了一个比较函数comp。
91. prev\_permutation函数：模板函数，有一个模板参数BidirectionalIterator；接受两个参数first和last；如果序列为空或只有一个元素，返回false，声明一个迭代器i指向最后一个元素，利用while true函数，声明一个迭代器ii等于i，--i，如果ii的值小于i的值，利用while循环找到第一个小于i的值的迭代器j，调用iter\_swap函数交换i和j，调用reverse函数将ii之后的元素逆转，返回true，如果i等于first，调用reverse函数将整个序列进行逆转，返回false；返回值类型为bool。
92. prev\_permutation函数：和前一个prev\_permutation函数类似，只是多了一个比较函数comp。
93. merge函数：模板函数，有三个模板参数InputIterator1，InputIterator2和OutputIterator；接受五个参数first1，last1，first2，last2和result；利用while循环将两个序列中较小的值移动到result中，循环结束后调用两个copy函数将两个序列中剩余的部分移动到result中；返回值类型为OutputIterator。
94. merge函数：和前一个merge函数类似，只是多了一个比较函数comp。
95. inplace\_merge函数：模板函数，有一个模板参数BidirectionalIterator；接受三个参数first，middle和last；如果两个区间有空区间，直接返回，否则调用inplace\_merge\_aux函数。
96. inplace\_merge\_aux函数：模板函数，有两个模板参数BidirectionalIterator和T；接受四个参数first，middle，last和T\*；调用distance函数算出两个区间的长度len1和len2，利用temporary\_buffer申请一个临时缓冲区buf，如果buffer.begin为空(内存分配失败)，调用merge\_without\_buffer函数，否则调用merge\_adaptive函数。
97. merge\_adaptive函数：模板函数，有三个模板参数BidirectionalIterator，Distance和Pointer；接受七个参数first，middle，last，len1，len2，buffer和buffer\_size；如果len1小于len2并且len1小于buffer\_size，调用copy函数将前半区间拷贝到缓冲区中并得到区间尾部buffer\_end，再调用merge函数进行区间合并；如果len2小于buffer\_size，调用copy函数将后半部分区间拷贝到缓冲区中，再调用merge\_backward函数进行合并；否则区间长度太长，需要分隔递归处理，声明first\_cut和second\_cut为first和middle，len11和len22都为0，如果len1大于len2，len11设为len1的一半，将first\_cut移动len11的长度，调用lower\_bound函数在第二个区间找到第一个大于等于first\_cut的值的元素并将结果赋给second\_cut，计算出middle到second\_cut的距离len22，如果len1小于等于len2，处理和前面类似，只是将lower\_bound函数换为了upper\_bound函数，调用rotate\_adaptive函数并将返回值设为new\_middle，递归调用merge\_adaptive函数对前后两半部分进行递归处理。
98. rotate\_adaptive函数：模板函数，有三个模板参数BidirectionalIterator1，BidirectionalIterator2和Distance；接受七个参数first，middle，last，len1，len2，buffer和buffer\_size；声明一个迭代器buffer\_end，如果len1大于len2并且len2小于等于buffer\_size，调用copy函数将后半段区间拷贝到buffer中并将返回值赋给buffer\_end，调用copy\_backward函数将前半段区间拷贝到后半段，调用copy函数将buffer中的元素拷贝到原区间的前半段；如果len1小于等于buffer\_size，进行类似的操作；否则调用rotate函数进行旋转；返回值类型为BidirectionalIterator1。
99. merge\_backward函数：模板函数，有三个模板参数BidirectionalIterator1，BidirectionalIterator2和BidirectionalIterator3；接受四个参数first1，last1，first2，last2和result；如果前半部分区间为空，调用copy\_backward函数将后半部分区间拷贝到result中，如果后半部分区间为空，调用copy\_backward函数将前半部分区间拷贝到result中，将last1和last2分别指向两个区间的最后一个元素，利用while true循环，如果last2的值小于last1的值，将last1的值赋给result，如果first1等于last1，调用copy\_backward函数将后半部分区间拷贝到result中并返回，移动last1；如果last2的值大于等于last1的值，将last2的值赋给result，如果first2等于last2，调用copy\_backward函数将前半部分区间拷贝到result中并返回，移动last1；返回值类型为BidirectionalIterator3。
100. merge\_without\_buffer函数：模板函数，有两个模板参数BidirectionalIterator和Distance；接受五个参数first，middle，last，len1和len2；如果两个空间中存在空区间，直接返回，如果两个区间总长度为2，如果第二个元素的值小于第一个元素的值，调用swap\_iter函数，直接返回；否则区间长度太长，需要分隔递归处理，声明first\_cut和second\_cut为first和middle，len11和len22都为0，如果len1大于len2，len11设为len1的一半，将first\_cut移动len11的长度，调用lower\_bound函数在第二个区间找到第一个大于等于first\_cut的值的元素并将结果赋给second\_cut，计算出middle到second\_cut的距离len22，如果len1小于等于len2，处理和前面类似，只是将lower\_bound函数换为了upper\_bound函数，调用rotate函数并将返回值设为new\_middle，递归调用merge\_without\_buffer函数对前后两半部分进行递归处理。
101. inplace\_merge函数：和前面的inplace\_merge函数类似，只是多了一个比较函数comp。
102. inplace\_merge\_aux函数：和前面的inplace\_merge\_aux函数类似，只是多了一个比较函数comp。
103. merge\_adaptive函数：和前面的merge\_adaptive函数类似，只是多了一个比较函数comp。
104. rotate\_adaptive函数：和前面的rotate\_adaptive函数类似，只是多了一个比较函数comp。
105. merge\_backward函数：和前面的merge\_backward函数类似，只是多了一个比较函数comp。
106. merge\_without\_buffer函数：和前面的merge\_without\_buffer函数类似，只是多了一个比较函数comp。
107. partial\_sort函数：模板函数，有一个模板参数RandomAccessIterator；接受三个参数first，middle和last；首先调用make\_heap函数对first到middle的元素建堆，利用for循环遍历middle到last的元素，如果当前元素的值小于first的值，调用\_\_pop\_heap函数将当前元素的值送入堆中，循环结束后调用sort\_heap进行排序。
108. partial\_sort函数：和前一个partial\_sort函数类似，只是多了一个比较函数comp。
109. partial\_sort\_copy函数：模板函数，有两个模板参数InputIterator和RandomAccessIterator；接受四个参数first，last，result\_first和result\_last；调用\_\_partial\_sort\_copy函数并返回；返回值类型为RandomAccessIterator。
110. \_\_partial\_sort\_copy函数：模板函数，有三个模板参数InputIterator，RandomAccessIterator和Distacne；接受五个参数first，last，result\_first，result\_last和Distance\*；如果结果区间为空，返回result\_last，声明一个迭代器result\_real\_last用于循环，利用while循环将first区间的值拷贝到result\_first区间中，调用make\_heap函数对result区间建堆，利用while循环遍历剩下的first区间元素，如果first的值小于result\_first的值，调用adjust\_heap函数将first传入并调整result堆，循环结束后调用sort\_heap函数进行排序，返回result\_real\_last；返回值类型为RandomAccessIterator。
111. partial\_sort\_copy函数：和前面的partial\_sort\_copy函数类似，只是多了一个比较函数comp。
112. \_\_partial\_sort\_copy函数：和前面的\_\_partial\_sort\_copy函数类似，只是多了一个比较函数comp。
113. partition函数：模板函数，有两个模板参数ForwardIterator和UnaryPredicate；接受三个参数first，last和unary\_pred；调用\_\_partition函数并返回；返回值类型为ForwardIterator。
114. \_\_partition函数(forward iterator版本)：模板函数，有两个模板参数ForwardIterator和UnaryPredicate；接受四个参数first，last，unary\_pred和forward\_iterator\_tag；如果序列为空，返回first，利用while循环寻找序列前端连续满足条件的元素，如果全部满足返回first，声明一个迭代器next等于first，利用while循环遍历剩下的元素，如果满足条件调用swap函数和first进行互换，循环结束后返回first；返回值类型为ForwardIterator。
115. \_\_partition函数(bidirectional iterator版本)：模板函数，有两个模板参数BidirectionalIterator和UnaryPredicate；接受四个参数first，last，unary\_pred和bidirecitional\_iterator\_tag；利用while true循环，利用while循环找到序列前端第一个不满足条件的元素，如果遍历结束直接返回first，利用while循环找到序列后端第一个满足条件的元素，如果遍历结束直接返回first，调用iter\_swap函数交换两个位置，循环结束后返回first；返回值类型为BidirectionalIterator。
116. stable\_partition函数：模板函数，有两个模板参数ForwardIterator和UnaryPredicate；接受三个参数first，last和unary\_pred；如果序列为空返回first，否则调用stable\_partition\_aux函数并返回；返回值类型为ForwardIterator。
117. stable\_partition\_aux函数：模板函数，有四个模板参数ForwardIterator，UnaryPredicate，T和Distance；接受五个参数first，last，unary\_pred，T\*和Distance\*；利用temporary\_buffer声明一个临时缓冲区buf，如果声明成功(buf.size()>0)，调用stable\_partition\_adaptive函数并返回，否则调用inplace\_stable\_partition函数并返回，返回值类型为ForwardIterator。
118. inplace\_stable\_partition函数：模板函数，有三个模板参数ForwardIterator，UnaryPredicate和Distance；接受四个参数first，last，unary\_pred和len；如果len为1，判断first的值是否满足条件，如果满足返回last否则返回first，把系列分为前后两段，分别递归调用inplace\_stable\_partition函数并将返回值调用rotate函数；返回值类型为ForwardIterator。
119. stable\_partition\_adaptive函数：模板函数，有四个模板参数ForwardIterator，Pointer，Predicate和Distance；接受六个参数first，last，pred，len，buffer和buffer\_size；如果len小于等于buffer\_size，声明两个指针result1和result2分别等于first和buffer，利用for循环遍历序列，如果当前元素满足条件将其拷贝到result中，否则拷贝到result2中，循环结束后将buffer中的元素调用copy函数拷贝到result1中，返回result1；如果len大于buffer\_size，将序列分为前后两部分，分别递归调用stable\_partition\_adaptive函数并将返回值调用rotate函数；返回值类型为ForwardIterator。
120. partition\_copy函数：模板函数，有四个模版参数InputIterator，OutputIterator1，OutputIterator2和UnaryPredicate；接受五个参数first，last，result\_true，result\_false和unary\_pred；利用for循环遍历序列，如果当前元素满足条件，将其拷贝到result\_true中，否则拷贝到result\_false中；返回值类型为pair<OutputIterator1，OutputIterator2>。
121. sort函数：模板函数，有一个模板参数RandomAccessIterator；接受两个参数first和last；如果不是空区间，先调用introsort\_loop函数，再调用final\_insertion\_sort函数。
122. introsort\_loop函数：模板函数，有两个模板参数RandomAccessIterator和Size；接受三个参数first，last和depth\_limit；利用while循环，当区间长度大于stl\_threshold(定义的一个全局变量)时，如果depth\_limit为0(当前递归深度过大)，调用partial\_sort(heap sort)并return，将depth\_limit减1，声明一个迭代器cut为调用unguarded\_partition函数(pivot采用three median方法确定)的返回值(后半区间的第一个元素)，对后半段区间递归调用introsort\_loop函数，将cut赋值给last(对前半段区间进行处理)。
123. unguarded\_partition函数：模板函数，有两个模板参数RandomAccessIterator和T；接受三个参数first，last和pivot；利用while true循环，利用while循环找到第一个大于等于pivot的元素，利用while循环找到第一个小于等于pivot的元素，如果first大于等于last，返回first，否则调用iter\_swap函数交换两个元素，调整first的值；返回值类型为RandomAccessIterator。
124. final\_insertion\_sort函数：模板函数，有一个模板参数RandomAccessIterator；接受两个参数first和last；如果区间长度大于stl\_threshold，对前stl\_threshold个元素调用insertion\_sort函数，对剩余的元素调用unguarded\_insertion\_sort函数(因为introsort\_loop函数得到了多个元素个数少于stl\_threshold的区间，所以最小的值一定落在前stl\_threshold个位置上)；否则对整个区间调用insertion\_sort函数。
125. insertion\_sort函数：模板函数，有一个模板参数RandomAccessIterator；接受两个参数first和last；如果区间为空直接return，利用for循环调用linear\_insert函数进行插入排序。
126. linear\_insert函数：模板函数，有一个模板参数RandomAccessIterator；接受两个参数first和last；令最后一个元素的值为value，如果value小于first的值，调用copy\_backward函数将区间整体后移一位，将value赋值给first，否则调用unguarded\_linear\_insert函数。
127. unguarded\_linear\_insert函数：模板函数，有两个模板参数RandomAccessIterator和T；接受两个参数last和value；利用while循环找到合适的位置并进行插入。
128. unguarded\_insertion\_sort函数：模板函数，有一个模板参数RandomAccessIterator；接受两个参数first和last；利用for循环对区间内元素调用unguarded\_linear\_insert函数。
129. lg函数：模板函数，有一个模板参数Size；接受一个参数n；利用for循环找到最大的使2的k次方小于等于n的值；返回值类型为Size。
130. sort函数：和前一个sort函数类似，只是多了一个比较函数comp。
131. introsort\_loop函数：和前一个introsort\_loop函数类似，只是多了一个比较函数comp。
132. unguarded\_partition函数：和前一个unguarded\_partition函数类似，只是多了一个比较函数comp。
133. final\_insertion\_sort函数：和前一个final\_insertion\_sort函数类似，只是多了一个比较函数comp。
134. insertion\_insert函数：和前一个insertion\_insert函数类似，只是多了一个比较函数comp。
135. linear\_sort函数：和前一个linear\_sort函数类似，只是多了一个比较函数comp。
136. unguarded\_linear\_sort函数：和前一个unguarded\_linear\_sort函数类似，只是多了一个比较函数comp。
137. unguarded\_insertion\_sort函数：和前一个unguarded\_insertion\_sort函数类似，只是多了一个比较函数comp。
138. stable\_sort函数：模板函数，有一个模板参数RandomAccessIterator；接受两个参数first和last；调用stable\_sort\_aux函数。
139. stable\_sort函数：模板函数，有三个模板参数RandomAccessIterator，T和Distance；接受四个参数first，last，T\*和Distance\*；利用temporary\_buffer申明一个临时缓冲区buf，如果申请失败，调用inplace\_stable\_sort函数，否则调用stable\_sort\_adaptive函数。
140. stable\_sort\_adaptive函数：模板函数，有三个模板参数RandomAccessIterator，Pointer和Distance；接受四个参数first，last，buffer和buffer\_size；计算区间长度的一半len(向上取整)，声明一个迭代器middle将区间分为前后两部分，如果len大于buffer\_size，对前后两部分区间递归调用stable\_sort\_adaptive函数，否则对前后两个区间分别调用merge\_sort\_with\_buffer函数，最后调用merge\_adaptive函数。
141. merge\_sort\_with\_buffer函数：模板函数，有三个模板参数RandomAccessIterator，Pointer和Distance；接受四个参数first，last，buffer和Distance\*；算出区间长度len，声明一个指针buffer\_last指向buffer的尾端，声明一个step\_size为stl\_chunk\_size(定义的一个全局变量)，调用chunk\_insertion\_sort函数，利用while循环，当step\_size小于len时，从first区间到buffer调用merge\_sort\_loop函数，将step\_size乘以2，从buffer区间到first区间调用merge\_sort\_loop函数，将step\_size乘以2。
142. chunk\_insertion\_sort函数：模板函数，有两个模板参数RandomAccessIterator和Distance；接受三个参数first，last和chunk\_size；利用while循环对每chunk\_size个元素调用insertion\_sort进行排序。
143. merge\_sort\_loop函数：模板函数，有三个模板参数RandomAccessIterator1，RandomAccessIterator2和Distance；接受四个参数first，last，result和step\_size；声明一个two\_step为step\_size的两倍，利用while循环，当区间长度大于等于two\_step时，调用merge函数将两个块进行合并，返回值赋给result，first移动two\_step，循环结束后，将step\_size设为剩余区间长度和step\_size的较小值，对剩余的两块调用merge函数进行合并。
144. inplace\_stable\_sort函数：模板函数，有一个模板参数RandomAccessIterator；接受两个参数first和last；如果区间长度小于15(不知道为什么设为15)，调用insertion\_sort函数并return，将区间分为前后两部分，分别递归调用inplace\_stable\_sort函数，再调用merge\_without\_buffer将前后两部分合并。
145. stable\_sort函数：和前一个stable\_sort函数类似，只是多了一个比较函数comp。
146. stable\_sort\_adaptive函数：和前一个stable\_sort\_adaptive函数类似，只是多了一个比较函数comp。
147. stable\_sort函数：和前一个stable\_sort函数类似，只是多了一个比较函数comp。
148. merge\_sort\_with\_buffer函数：和前一个merge\_sort\_with\_buffer函数类似，只是多了一个比较函数comp。
149. chunk\_insertion\_sort函数：和前一个chunk\_insertion\_sort函数类似，只是多了一个比较函数comp。
150. merge\_sort\_loop函数：和前一个merge\_sort\_loop函数类似，只是多了一个比较函数comp。
151. inplace\_stable\_sort函数：和前一个inplace\_stable\_sort函数类似，只是多了一个比较函数comp。
152. mergesort函数：模板函数，有一个模板参数BidirectionalIterator；接受两个参数first和last；算出区间长度n，如果n为0或1，直接return，否则将区间分为两半并递归调用mergesort函数，然后用inplace\_merge函数将前后两部分合并起来。
153. nth\_element函数：模板函数，有一个模板参数RandomAccessIterator；接受三个参数first，nth和last；利用while循环，当区间长度大于3时，调用unguarded\_partition函数进行划分，并将返回值赋给迭代器cut，如果nth落在后半区间，在后半区间继续划分(令first等于cut)，否则在前半区间继续划分(令last等于cut)，循环结束后调用insertion\_sort函数进行插入排序。(unguarded\_partition函数的作用之一就是划分后的前半部分区间(假设长度为k)就是最小的前k的数)。
154. nth\_element函数：和前一个nth\_element函数类似，只是多了一个比较函数comp。
155. unique\_copy函数：模板函数，有两个模板参数InputIterator和OutputIterator；接受三个参数first，last和result；如果区间为空，返回result，调用\_\_unique\_copy函数并返回；返回值类型为OutputIterator。
156. \_\_unique\_copy函数(forward iterator版本)：模板函数，有两个模板参数InputIterator和OutputIterator；接受四个参数first，last，result和forward\_iterator\_tag；将第一个元素拷贝到result中，利用while循环遍历所有元素，如果当前元素不等于result的值，将其拷贝到result中，循环结束返回++result；返回值类型为ForwardIterator。
157. \_\_unique\_copy函数(output iterator版本)：模板函数，有两个模板参数InputIterator和OutputIterator；接受四个参数first，last，result和output\_iterator\_tag；将first的值声明为value，将value赋给result，利用while循环遍历所有元素，如果value不等于当前元素的值，令value为当前元素的值，并将value拷贝到result中，循环结束后返回++result；返回值类型为OutputIterator。
158. unique\_copy函数：和前一个unique\_copy函数类似，只是多了一个比较函数comp。
159. \_\_unique\_copy函数(forward iterator版本)：和之前的forward iterator版本的\_\_unique\_copy函数类似，只是多了一个比较函数comp。
160. \_\_unique\_copy函数(output iterator版本)：和之前output iterator版本的\_\_unique\_copy函数类似，只是多了一个比较函数comp。
161. unique函数：模板函数，有一个模板参数ForwardIterator；接受两个参数first和last；调用adjacent\_find函数找到第一个重复的位置并赋给first，调用unique\_copy函数处理剩下的元素并返回；返回值类型为ForwardIterator。
162. unique函数：和前一个unique函数类似，只是多了一个比较函数comp。

# 5 仿函数

1. 函数对象(在文件functional.h中)
2. unarg\_function：模板类，有两个模板参数Arg和Result；包括两个typedef(argument\_type和result\_type)。
3. binary\_function：模板类，有三个模板参数Arg1，Arg2和Result；包括三个typedef(first\_argument\_type，second\_argument\_type和result\_type)。
4. plus：模板类，继承binary\_function，有一个模板参数T；重载operator()实现加法。
5. minus：模板类，继承binary\_function，有一个模板参数T；重载operator()实现减法。
6. multiplies：模板类，继承binary\_function，有一个模板参数T；重载operator()实现乘法。
7. divides：模板类，继承binary\_function，有一个模板参数T；重载operator()实现除法。
8. modulus：模板类，继承binary\_function，有一个模板参数T；重载operator()实现取模。
9. negate：模板类，继承unary\_function，有一个模板参数T；重载operator()实现取反。
10. identity\_element函数(inline函数)：加法的证同元素；模板函数，有一个模板参数T；接受一个参数(类型为plus<T>)，将0转换为T类型后返回；返回值类型为T。
11. identity\_element函数(inline函数)：乘法的证同元素；模板函数，有一个模板参数T；接受一个参数(类型为multiplies<T>)，将1转换为T类型后返回；返回值类型为T。
12. equal\_to：模板类，继承binary\_function，有一个模板参数T；重载operator()判断两个参数是否相等。
13. not\_equal\_to：模板类，继承binary\_function，有一个模板参数T；重载operator()判断两个参数是否不相等。
14. greater：模板类，继承binary\_function，有一个模板参数T；重载operator()判断第一个参数是否大于第二个参数。
15. less：模板类，继承binary\_function，有一个模板参数T；重载operator()判断第一个参数是否小于第二个参数。
16. greater\_equal：模板类，继承binary\_function，有一个模板参数T；重载operator()判断第一个参数是否大于等于第二个参数。
17. less\_equal：模板类，继承binary\_function，有一个模板参数T；重载operator()判断第一个参数是否小于等于第二个参数。
18. logical\_and：模板类，继承binary\_function，有一个模板参数T；重载operator()实现逻辑与。
19. logical\_or：模板类，继承binary\_function，有一个模板参数T；重载operator()实现逻辑或。
20. logical\_not：模板类，继承unary\_function，有一个模板参数T；重载operator()实现逻辑非。
21. identity：模板类，继承unary\_function，有一个模板参数T；重载operator()返回参数自身。
22. Select1st：模板类，继承unary\_function，有一个模板参数Pair；重载operator()返回参数的第一个元素(注意参数类型)。
23. Select2nd：模板类，继承unary\_function，有一个模板参数Pair；重载operator()返回参数的第二个元素(注意参数类型)。
24. unary\_negate：模板类，继承unary\_function，有一个模板参数Predicate；protected属性下有一个成员pred；构造函数接受一个参数rhs(Predicate类型)，将rhs赋给pred；重载operator()返回pred的否定值。
25. not1：辅助函数，方便使用unary\_negate；模板函数，有一个模板参数Predicate，接受一个参数pred；返回unary\_negate对象；返回值类型为unary\_negate。
26. binary\_negate：模板类，继承binary\_function，有一个模板参数Predicate；protected属性下有一个成员pred；构造函数接受一个参数rhs(Predicate类型)，将rhs赋给pred；重载operator()返回pred的否定值。
27. not2：辅助函数，方便使用binary\_pred；模板函数，有一个模板参数Predicate；接受一个参数pred；返回binary\_negate对象；返回值类型为binary\_negate。
28. binder1st：模板类，继承unary\_function，有一个模板参数Operation；protected属性下有两个变量op和 value(Operation的first\_argument\_type)；构造函数接受两个参数x(Operation)和y(first\_argument\_type)，进行相应的赋值；重载operator()，对value和参数x调用op函数。
29. bind1st：辅助函数，更方便使用binder1st；模板函数，有两个模板参数Operation和T；接受两个参数fn和x；返回binder1st对象；返回值类型为binder1st。
30. binder2nd：和binder1st类似，只是将第二个参数绑定为参数。
31. bind2nd：和bind1st类似，只是将第二个参数绑定为参数。
32. unary\_compose：模板类，继承unary\_function，有两个模板参数Operation1和Operation2；protected属性下维护两个变量fn1和fn2；构造函数接受两个参数x(Operation1)和y(Operation2)，进行相应的赋值；重载operator()实现Operator1(Operator2(x))的操作。
33. compose1：辅助函数，更方便使用unary\_compose；模板函数，有两个模板参数Operation1和Operation2；接受两个参数fn1和fn2；返回unary\_compose对象；返回值类型为unary\_compose。
34. binary\_compose：模板类，继承unary\_function；有三个模板参数Operation1，Operation2和Operation3；protected属性下维护三个变量fn1，fn2和fn3；构造函数接受三个参数x，y和z，进行相应的赋值；重载operator()实现Operator1(Operator2(x)，Operator3(x))的操作。
35. compose2：辅助函数，更方便使用binary\_compose；模板函数，有三个模板参数Operation1，Operation2和Operator3；接受三个参数fn1，fn2和fn3；返回binary\_compose对象；返回值类型为binary\_compose。
36. pointer\_to\_unary\_function：模板类，继承unary\_function；有两个模板参数Arg和Result；protected属性下维护一个函数指针ptr；有一个默认构造函数；构造函数接受一个参数 rhs(函数指针)，进行相应的赋值；重载operator()调用ptr指针实现函数调用。
37. ptr\_fun：辅助函数，更方便使用pointer\_to\_unary\_function；模板函数，有两个模板参数Arg和Result；接受一个参数x(函数指针)；返回pointer\_to\_unary\_function对象；返回值类型为pointer\_to\_unary\_function。
38. pointer\_to\_binary\_function：模板类，继承binary\_function；有三个模板参数Arg1，Arg2和Result；protected属性下维护一个函数指针ptr；有一个默认构造函数；构造函数接受一个参数x(函数指针)，进行相应的赋值；重载operator()调用ptr指针实现函数调用。
39. ptr\_fun：辅助函数，更方便使用pointer\_to\_binary\_function；模板函数，有三个模板参数Arg1，Arg2和Result；接受一个参数x(函数指针)；返回pointer\_to\_binary\_function对象；返回值类型为pointer\_to\_binary\_function。
40. mem\_fun\_t(无参数，通过指针调用，non-const成员函数)：模板类，继承unary\_function，有两个模板参数Ret和T；private属性下有一个成员(T类下的成员函数指针)；构造函数有一个参数，成员函数指针pf，进行相应的赋值；重载operator()函数通过传入的参数p(T类型)调用成员函数。
41. const\_mem\_fun\_t(无参数，通过指针调用，const成员函数)：模板类，继承unary\_function；有两个模板参数Ret和T；private属性下有一个成员(T类下的成员函数指针，const类型成员函数)；构造函数有一个参数，成员函数指针pf，进行相应的赋值；重载operator()函数通过传入的参数p(T类型)调用成员函数。
42. mem\_fun\_ref\_t(无参数，通过引用调用，non-const成员函数)：和mem\_fun\_t类似，在重载operator()上调用成员函数方式上有所不同。
43. const\_mem\_fun\_ref\_t(无参数，通过引用调用，const成员函数)：和const\_mem\_fun\_t类似，在重载operator()上调用成员函数方式上有所不同。
44. mem\_fun1\_t(有一个参数，通过指针调用，non-const成员函数)：模板类，继承binary\_function，有三个模板参数Ret，T和Arg；private属性下有一个成员(T类下的成员函数指针)；构造函数有一个参数，成员函数指针pf，进行相应的赋值；重载operator()函数通过传入的参数p(T类型)调用成员函数。
45. const\_mem\_fun1\_t(有一个参数，通过指针调用，const成员函数)：和mem\_fun1\_t类似，只是维护的成员为const成员指针。
46. mem\_fun1\_ref\_t(有一个参数，通过引用调用，non-const成员函数)：和mem\_fun1\_t类似，在重载operator()上调用成员函数方式上有所不同。
47. const\_mem\_fun1\_ref\_t(有一个参数，通过引用参数，const成员函数)：和const\_mem\_fun1\_t类似，在重载operator()上调用成员函数方式上有所不同。
48. mem\_fun：辅助函数；模板函数，有两个模板参数Ret和T；接受一个参数f(成员函数指针)；返回mem\_fun\_t的对象；返回值类型为mem\_fun\_t。
49. mem\_fun：辅助函数；模板函数，有两个模板参数Ret和T；接受一个参数f(成员函数指针)；返回const\_mem\_fun\_t的对象；返回值类型为const\_mem\_fun\_t。
50. mem\_fun\_ref：辅助函数；模板函数，有两个模板参数Ret和T；接受一个参数f(成员函数指针)；返回mem\_fun\_ref\_t的对象；返回值类型为mem\_fun\_ref\_t。
51. mem\_fun\_ref：辅助函数；模板函数，有两个模板参数Ret和T；接受一个参数f(成员函数指针)；返回const\_mem\_fun\_ref\_t的对象；返回值类型为const\_mem\_fun\_ref\_t。
52. mem\_fun：辅助函数；模板函数，有两个模板参数Ret和T；接受一个参数f(成员函数指针)；返回mem\_fun1\_t的对象；返回值类型为mem\_fun1\_t。
53. mem\_fun：辅助函数；模板函数，有两个模板参数Ret和T；接受一个参数f(成员函数指针)；返回const\_mem\_fun1\_t的对象；返回值类型为const\_mem\_fun1\_t。
54. mem\_fun\_ref：辅助函数；模板函数，有两个模板参数Ret和T；接受一个参数f(成员函数指针)；返回mem\_fun1\_ref\_t的对象；返回值类型为mem\_fun1\_ref\_t。
55. mem\_fun\_ref：辅助函数；模板函数，有两个模板参数Ret和T；接受一个参数f(成员函数指针)；返回const\_mem\_fun1\_ref\_t的对象；返回值类型为const\_mem\_fun1\_ref\_t。
56. mem\_fun1：辅助函数；模板函数，有两个模板参数Ret和T；接受一个参数f(成员函数指针)；返回mem\_fun1\_t的对象；返回值类型为mem\_fun1\_t。
57. mem\_fun1：辅助函数；模板函数，有两个模板参数Ret和T；接受一个参数f(成员函数指针)；返回const\_mem\_fun1\_t的对象；返回值类型为const\_mem\_fun1\_t。
58. mem\_fun1\_ref：辅助函数；模板函数，有两个模板参数Ret和T；接受一个参数f(成员函数指针)；返回mem\_fun1\_ref\_t的对象；返回值类型为mem\_fun1\_ref\_t。
59. mem\_fun1\_ref：辅助函数；模板函数，有两个模板参数Ret和T；接受一个参数f(成员函数指针)；返回const\_mem\_fun1\_ref\_t的对象；返回值类型为const\_mem\_fun1\_ref\_t。

# 6 适配器

1. 应用于容器

Queue和stack都是修饰deque的接口形成的容器，属于应用于容器的适配器。

1. 应用于迭代器

insert iterator，reverse iterator，iostream iterator都是应用于迭代器的例子。

1. 应用于仿函数

配接操作包括bind，negete，compose和对一般函数或成员函数的修饰(使其成为一个仿函数)。