# STL六大部件

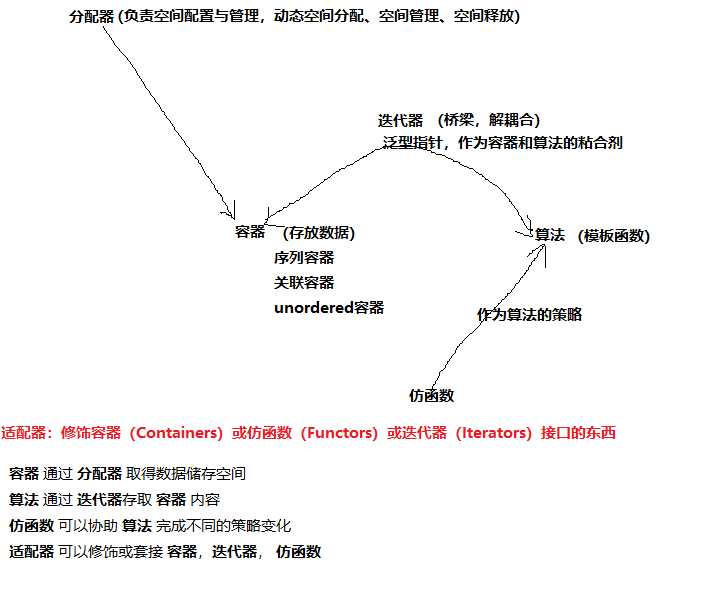
## STL简介

泛型编程generic programming

将算法和数据结构分别实现：大部分算法被抽象，被泛化，独立于数据结构

模板和偏特化是实现STL的核心

程序尽可能通用



算法看不见容器，它所需要的一切信息都要从迭代器中取得，而迭代器必须能够回答算法的所有提问，才能搭配该算法的所有操作。

# STL预备知识

## 区间：前闭后开 [ )

c.begin() c.end()：执行容器最后一个元素的下一个元素

## 遍历容器中的每个元素

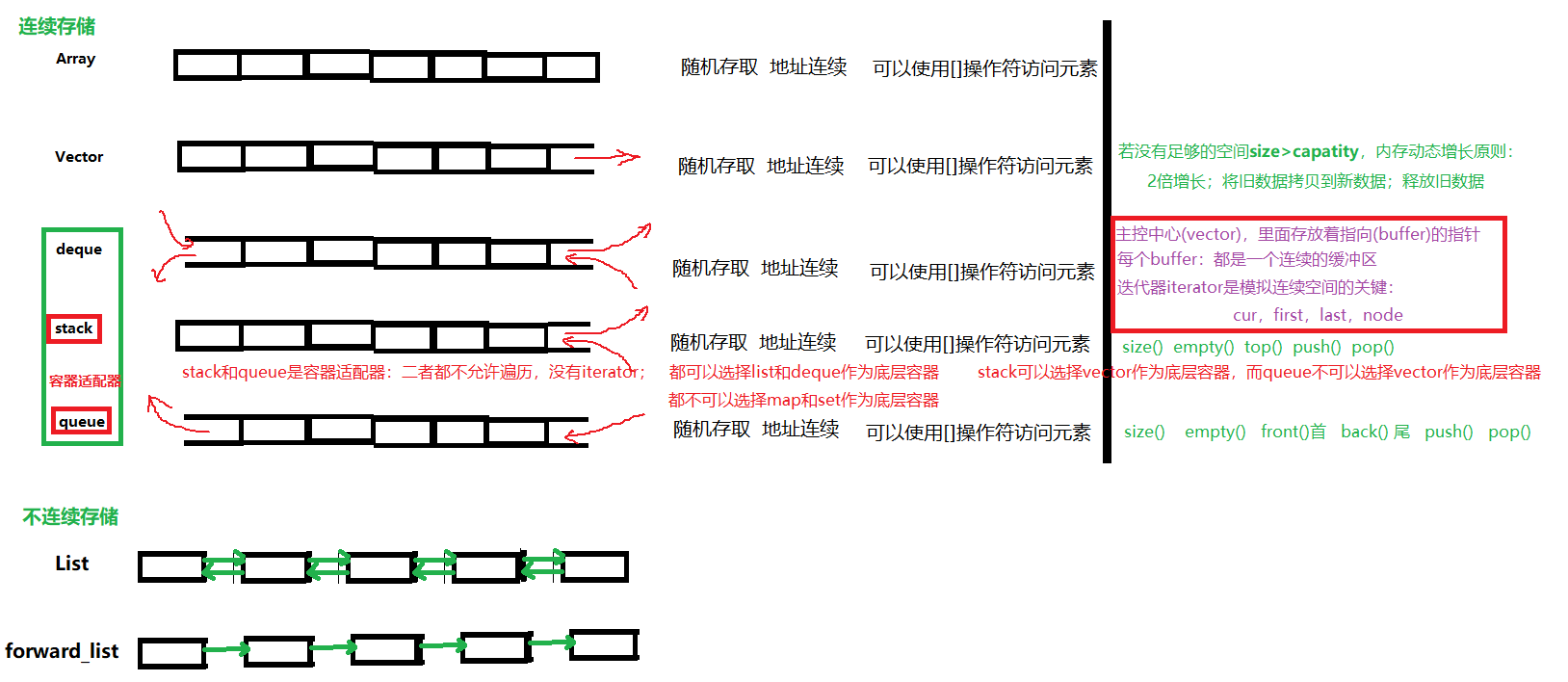
方式一：for(Container<type>::iterator iter = C.begin(); iter!=C.end;iter++)  **🡪 \*iter**

方式二：auto关键字——类型推导

For(auto elem : Containers) **🡪 elem**

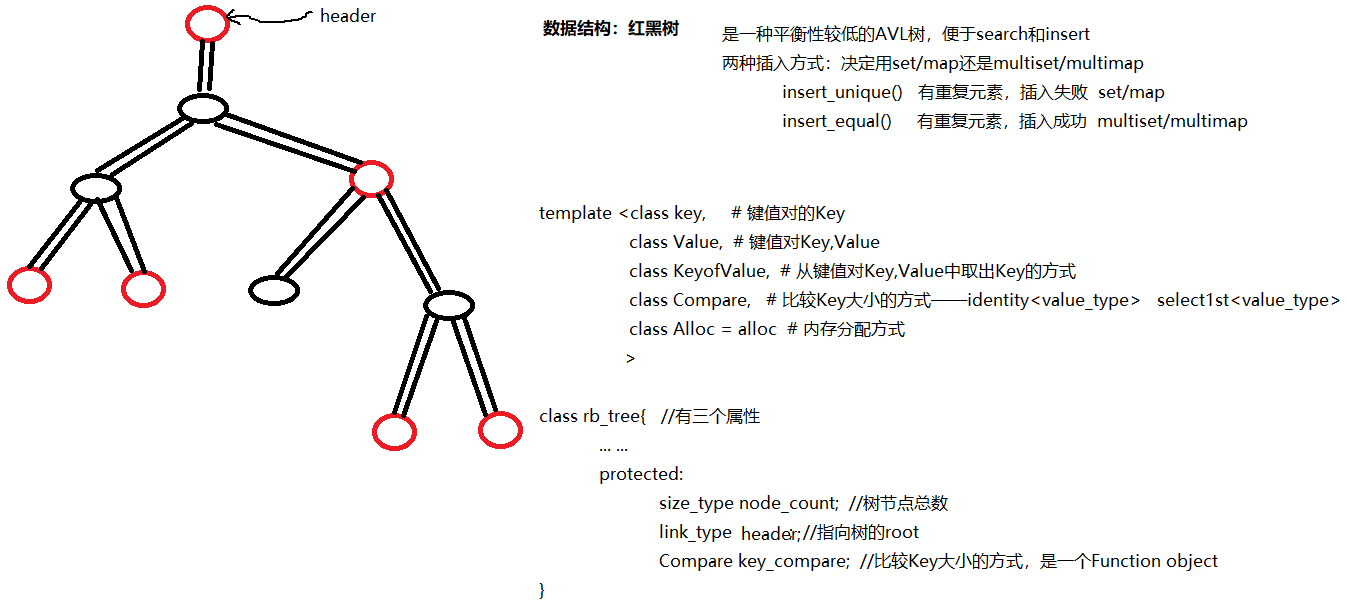
# 容器Containers

## （1）序列容器



## （2）关联容器（rb\_tree）

红黑树的特性：按照iterator方式遍历，是排好序的🡪**“元素自动排序”**



### 1-红黑树的性能分析： 查找/插入/删除：log(N)

set/multiset不能使用迭代器更改key的值

map/multimap不能使用迭代器更改key的值，但是可以更改value的值

### 2-介绍map的[]特殊的操作符：map[key] = value

如果key存在，则返回key对应的value

如果key不存在，则将(key,value)键值对安插到map中

### 3-Map的insert

mapStudent.insert(pair<int, string>(1, "student\_one"));

mapStudent[1] = "student\_one";

### 4-Map的反向迭代器：reserver\_iterator rbegin() rend() ->first ->second

1. **map<int, string>::reverse\_iterator** iter;
2. **for(iter = mapStudent.rbegin(); iter != mapStudent.rend(); iter++)**
3. cout<<iter->first<<" + "<<iter->second<<endl;

### 5-Map的三种数据查找元素，并返回迭代器位置： 迭代器=find(key)

1. map<int, string>::iterator iter = mapStudent.find(1);

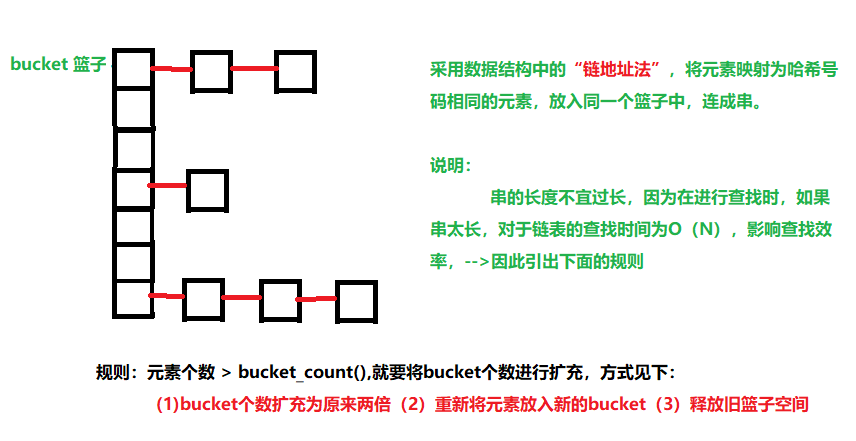
### 6-Map的erase方法

Iterator erase(iterator iter) //删除iter指向的元素，返回值Iterator指向iter的后继

Iterator erase(iterator first, iterator last) //删除一个范围：前开后闭[first,last）

size\_type erase(const Key& key) //通过关键字删除，返回值是0或1

## （3）Unordered容器（HashTable）无序



# 面试题型1：简述STL概念（非常重要）

答：

（1）

泛型编程；

程序通用性；

（2）

六大部件的简介和交互（特别是迭代器将容器[数据]和算法[操作]解耦）

# 面试题型2：红黑树 哈希表

## 红黑树的特性与其在C++ STL中的应用

关联容器：set,multiset map,multimap

## 2.map和set的3个问题。

### （1）为何map和set的插入删除效率比用其他序列容器vector，deque高？

因为对于关联容器来说，底层实现是红黑树，插入删除时，**不需要做内存拷贝和内存移动**。

### **（2）**为何map和set每次Insert之后，以前保存的iterator不会失效？

因为插入操作只是结点指针换来换去，结点内存没有改变。而iterator就像指向结点的指针，内存没变，指向内存的指针也不会变。

### ****（3）当数据元素增多时（从10000到20000），map的set的查找速度会怎样变化？****

RB-TREE用二分查找法，时间复杂度为logn，所以从10000增到20000时，查找次数从log10000=14次到log20000=15次，多了1次而已。

## STL 源码中的 hash table的实现

## STL 的 map 和 unordered \_map 的区别

(1) map

内部实现了一个**红黑树**，

元素是**自动排序**

查找，插入，删除时间复杂度O(logN)

(2) unordered\_map

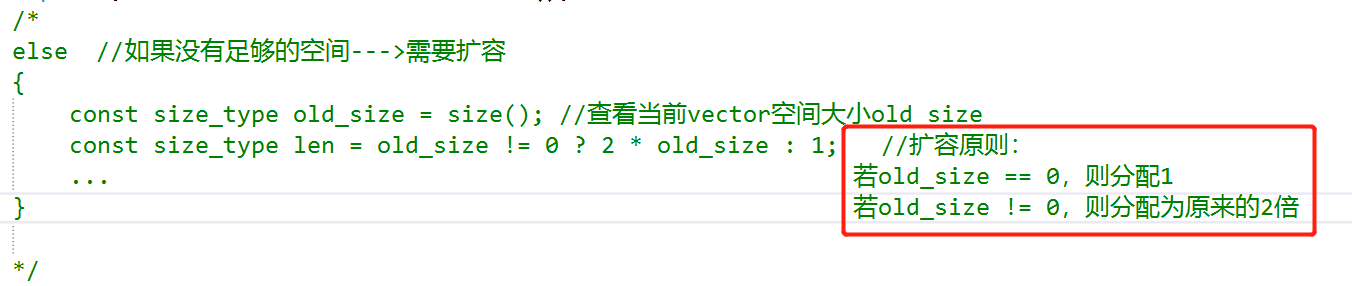
内部实现了一个**哈希表**

元素是**无序的**

查找时间复杂度O(1)

# 面试题型3：Vector考点

## Vector内存分配与扩容（vector的扩容机制，问到了源码层次）

先申请一定的大小的数组, 当数组填满之后,另外申请一块原数组两倍大的新数组, 然后把原数组的数据拷贝到新数组, 最后释放原数组的大小。   


## Vecotr 和List 的区别和使用场景

**区别**

动态数组；双向链表

存取方式：随机存取，通过索引查找

连续存储：

vector在进行插入和删除时，有元素的移动，时间复杂度为O(N)

list可以进行高效的插入和删除

当vector空间不够时，需要重新申请一块内存空间

**插入和删除数据：**

vector需要对现有数据进行复制移动，如果vector存储的对象很大或者构造函数很复杂，则开销较大，如果是简单的小数据，效率优于list

list需要对现有数据进行遍历，但在首部和尾部插入和删除数据，效率很高。

**使用场景**

高效的随机存取，而不在乎插入和删除的效率，使用vector;

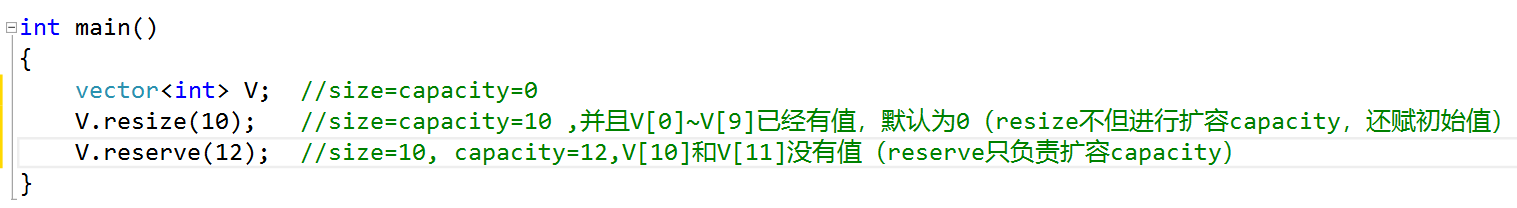
大量的插入和删除，而不关心随机存取，则应使用list。

## 3. Vector存入大量的数据，两种方式性能对比

(1)不断的push\_back():不断的扩容，需要内存申请，拷贝，释放，消耗CPU

(2)先申请预备空间V.reserve(capacity)在进行push\_back

**知识补充：vector的函数resize和reserve二者的区别**



## 强制释放vector的缓冲区——swap技法（effective STL）

由于vector的内存占用空间capacity只增不减，比如你首先分配了10,000个字节，然后erase掉后面9,999个，留下一个有效元素，但是内存占用capacity仍为10,000个。

所有内存空间是在vector析构时候才能被系统回收。empty()用来检测容器是否存在元素；clear()可以清空所有元素，但是即使clear()，vector所占用的内存空间capacity依然不变，无法保证内存的回收。

vector<int> V;

vector<int>().swap(V); //或者V.swap(vector<int> ())

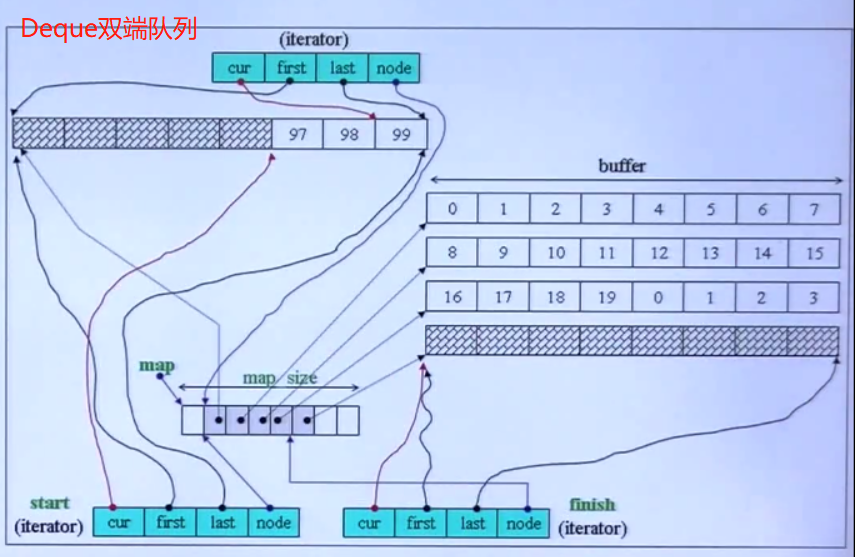
经过swap()后，size=capacity=0

解释：

swap技法就是通过交换函数swap()，使vector离开自身的作用域，从而强制释放vector的内存空间。

# 面试题型4：双端队列

**双端队列的的数据的存储形式：主控中心，缓冲区buffer**



**Question：令人疑惑的是——为什么上图中3个迭代器类型的指针？**

**Answer：**

**每个容器都有begin()和end()函数，指向首部和尾部；因此begin()调用start.cur指针，end()调用finish.cur指针；**

**另外一个迭代器（是用于遍历deque的迭代器） ：iterator有四个成员**

T\* cur; // 指向当前缓冲区中的元素

T\* first; // 当前缓冲区的起点

T\* last; // 当前缓冲区的终点

map\_pointer node; // 指向管控中心，用于越界时，进行跳转

**Deque**

(1) 主控中心:中的元素是指针，指向每一个缓冲区buffer

(2) 维护着start和finish两个迭代器，分别指向deque的首尾

(3) 内存分配：当内存不够时，主控中心向左或向右增加一个元素（该元素是指针），指针再指向一个新开辟一个buffer。

# 面试题型5：C++ STL 迭代器失效问题

## 1.对于vector和string的插入和删除：

**（插入元素时）：需要考虑：内存是否被重新分配**

如果容器内存被重新分配，iterators,pointers,references失效；

**（插入删除元素时）**

插入删除点之前的iterator有效，插入删除点之后的iterator失效；

## 2.对于list和forward\_list的插入和删除：

所有的iterator,pointer和refercnce都有效

总结：迭代器失效与否由容器的底层的存储结构决定：

顺序存储：vector，deque

重新分配内存空间，全部失效

It之前的不失效，it之后的失效

不连续存储：list，forward\_list，map，set

迭代器，不失效

## 补充1：为什么vector的插入操作可能会导致迭代器失效？

当vector动态增加大小时，如果capacity不够时，需要额外申请空间；将旧空间的数据全部拷贝到新空间中；释放旧空间。🡪因此，vector会导致迭代器失效

## ****补充2：vector、deque、list、map用erase（it）后，迭代器的变化。****

vector和deque是序列式容器，其内存分别是连续空间和分段连续空间，删除迭代器it后，伴随着数据的移动和复制，t前面的迭代器不失效，但是后面的迭代器都失效了，此时it指向被删除元素的下一个元素。

List是双向链表存储结构，地址不连续，删除迭代器it后，仅仅是指针发生变动，对其他迭代器不影响，因此迭代器都不会失效

Map使用红黑树，删除迭代器it，迭代器依然有效。

# 面试题型6：list.sort()和container.sort()原理以及性能对比

（1）STL容器alogrithm的排序，支持随机访问的容器：vector,deque,string

**原理：快速排序，时间复杂度：O(N\*logN)**

1. **template <class RandomAccessIterator>**
2. **void sort ( RandomAccessIterator first, RandomAccessIterator last );**
4. **template <class RandomAccessIterator, class Compare>**
5. **void sort ( RandomAccessIterator first, RandomAccessIterator last, Compare comp );**

STL容器的alogrithm的sort()函数要求：

1. **随机迭代器**，能用此算法的容器是支持随机访问的容器：vector, deque, string。  
2.  要求输入一个范围[first, last)  
3. 第一个版本默认使用operator<进行比较，默认升序排序；第二个版本可以使用自定义的comp规则进行比较.

（2）成员list排序调用自带的list::sort，不能调用STL::sort()函数

**原理：归并排序，时间复杂度：O(N\*logN)**

# 面试题型7：traits萃取编程（模板，偏特化）

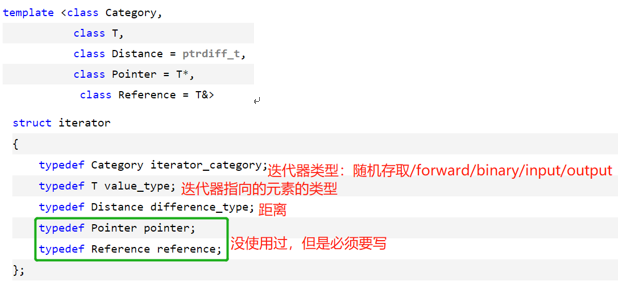
## 1. 功能：简单的说——判断出正确的参数类型

iterator通过traits，实现alogrithm与container的交互：

alogrithm通过向iterator提问，iterator必须能够回答alogrithm所提出的问题，才能搭配该alogrithm的进行操作。

在此过程中，Traits起到了回答alogrithm的问题的作用。

那么，问题来了，alogrithm到底提出了什么问题呢？其实很简单——algorithm只是在问iterator五个问题，分别是：



再次说明：iterator必须能够回答这五个alogrithm问题，alogrithm才能使用算法去处理iterator。

## 如何使用traits指定类型：

**方式1：内嵌类型**

1. **template** <**class** Iterator>
2. **struct** iterator\_traits {
3. **typedef** **typename** Iterator::value\_type value\_type;//声明内嵌类型
4. ...
5. };
7. **template** <**class** Iterator>
8. **typename** **iterator\_traits<Iterator>::value\_type**  GetValue(Iterator iter)
9. {
10. **return** \*iter;
11. }

代码解读：

typename：告诉编译器，typename后面跟的一堆东西是一个数据类型

**iterator\_traits<Iterator>::value\_type**：整体表示模板函数的返回类型，其中iterator\_traits<Iterator>::等价于ClassName<TypeName>::，表示使用模板类型为TypeName，类ClassName的成员属性value\_type 🡪 因此返回值类型为**iterator\_traits<Iterator>::value\_type。**

**说明：**

以上这种方法对**原生指针**不可行，所以iterator\_traits针对原生指针的一个版本就应运而生。下面是针对\*Tp和const \*Tp的版本，也称为iterator\_traits版本的偏特化。iterator\_traits的偏特化版本解决了原生指针的问题。🡪🡪🡪现在不管迭代器是自定义类模板, 还是原生指针(Tp\*, const Tp\*),struct iterator\_traits都能萃取出正确的value type类型。

**方式2：原生指针——模板偏特化（针对于指针类型和const指针类型）**

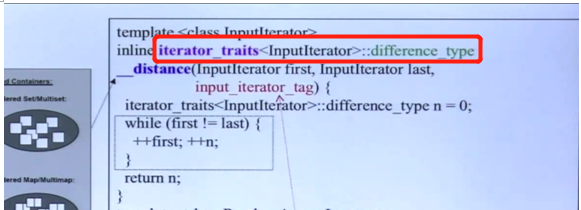
1. **template** <**class** Tp>
2. **struct** iterator\_traits<Tp\*> {
3. **typedef** Tp  value\_type;
4. ...
5. };
7. **template** <**class** Tp>
8. **struct** iterator\_traits<**const** Tp\*> {
9. **typedef** Tp  value\_type;
10. ...
11. };

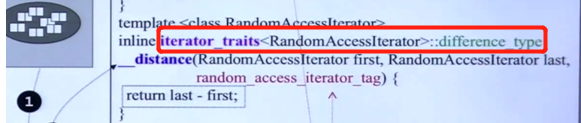
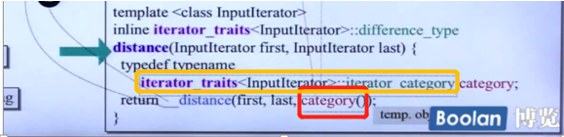
**总结：**

之所以要萃取（Traits）迭代器相关的类型，就是要把迭代器相关的类型用于**声明局部变量**、**用作函数的返回值**等一系列行为。**对于原生指针和point-to-const类型的指针，采用模板偏特化技术对其进行特殊处理**。

1. 要使用Traits功能，则必须自行以内嵌型别定义的方式定义出相应型别。

## 3.代码举例：分析traits的使用



上面三个图的代码分析：distance(first,last)函数

(1)distance(first,last)🡪调用\_\_distance(first,last,**category()**)，其中category()的类型是通过Traits获得的，即

(2)得到category的类型后，用category()创建一个临时对象，去调用\_\_distance(first,last,\_\_\_\_\_\_),其中\_\_\_\_\_\_参数有两种形式：input\_iterator\_tag和random\_access\_iterator\_tag

(3) category()与” input\_iterator\_tag和random\_access\_iterator\_tag”，进行类型匹配（本质上是函数重载），匹配成功后，调用匹配的函数，执行操作

(4)返回值类型，同理，也是通过Traits进行获取

## 4.STL约定

自定义的迭代器，必须继承std::iterator和iterator::traits，才能让它融入STL的大家庭中，无缝的使用各种泛型alogrithm。

# 面试题型8：仿函数（为alogrithm服务）类模板

说的通俗点就是在一个类中利用运算符重载重载"()"，让类拥有函数的使用特性和功能，这个class创建出来的对象，就是函数对象（它是一个对象，但是像一个函数，因此叫做仿函数）

## 举例：一个简单的仿函数

1. **template**<**class** T>
2. **struct** A  //一个类
3. {
4. **bool** operator()(**const** T& a,**const** T& b)  //重载()运算符
5. {
6. **return** (a == b);
7. }
8. };
10. **int** main()
11. {
12. A<**int**> aa;  //创建一个类对象aa
13. cout << aa(1, 2) << endl;  //类对象aa，调用()函数
14. **return** 0;
15. }

## 可配接（adaptable）的关键（继承两个类）

1.仿函数的相应型别主要用来表现函数参数型别和传回值**型别**

2. 任何仿函数，只有继承下面其中一个class，才拥有了那些相应**型别**，也就自动拥有了配接能力，能够被适配器改造。

template <class Arg, class Result>

struct unary\_function **//一个操作数：如，取负值**

{

typedef Arg argument\_type;

typedef Result result\_type;

};

template <class Arg1, class Arg2, class Result>

struct binary\_function **//两个操作数：如，相加，相与，比较大小**

{

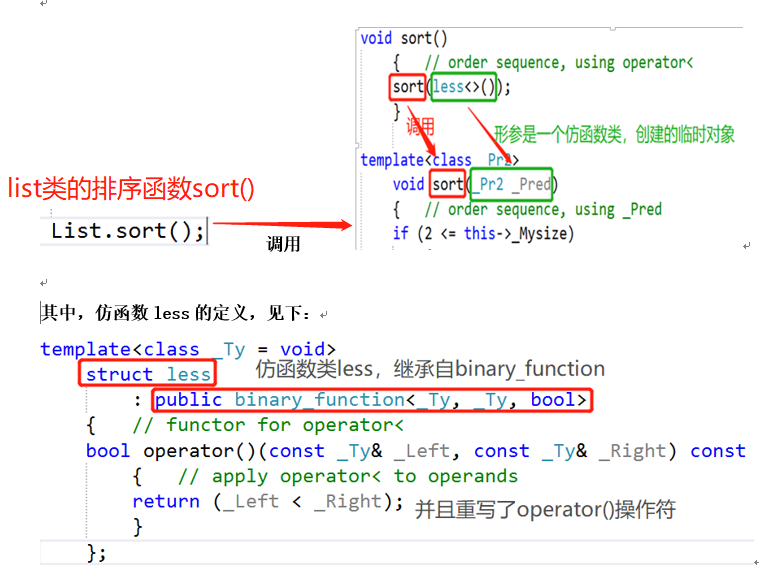
typedef Arg1 first\_argument\_type;

typedef Arg2 second\_argument\_type;

typedef Result result\_type;

};

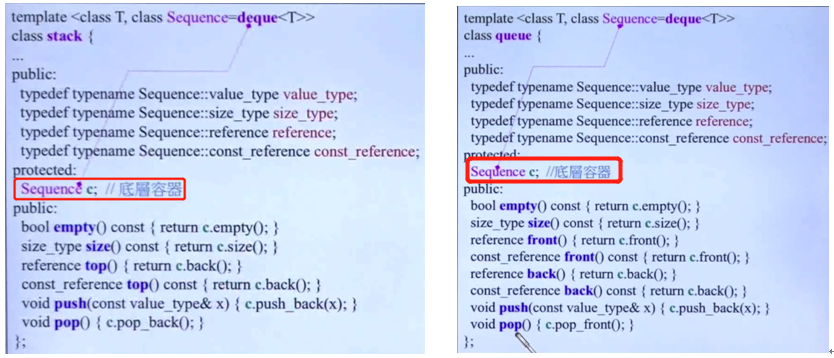
**举例说明：List双向列表的sort()排序方法，执行过程种，使用仿函数less<>()，详细见下：**



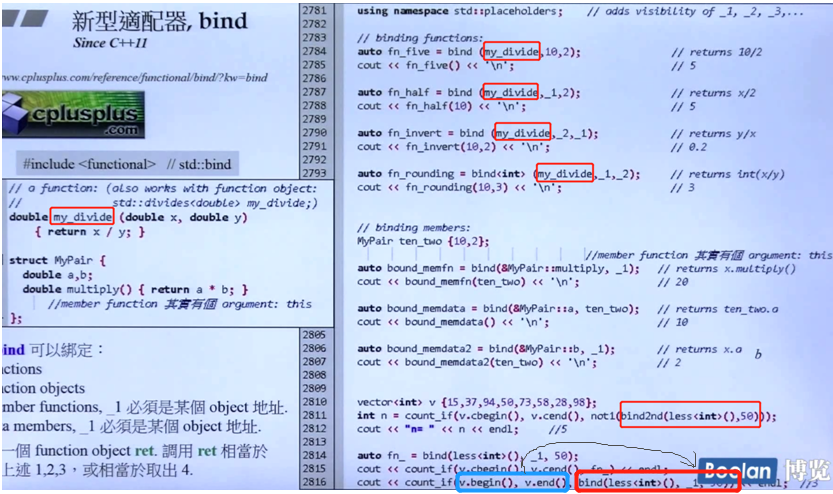
# 面试题型9：适配器（3种）

## (1)容器适配器：deque🡪stack,queue

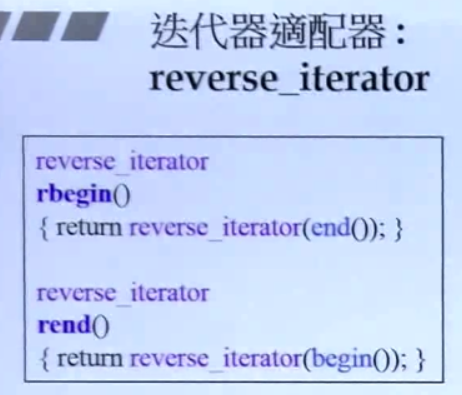
本质：Stack和queue使用deque作为底层容器，所有的操作都借助deque去实现



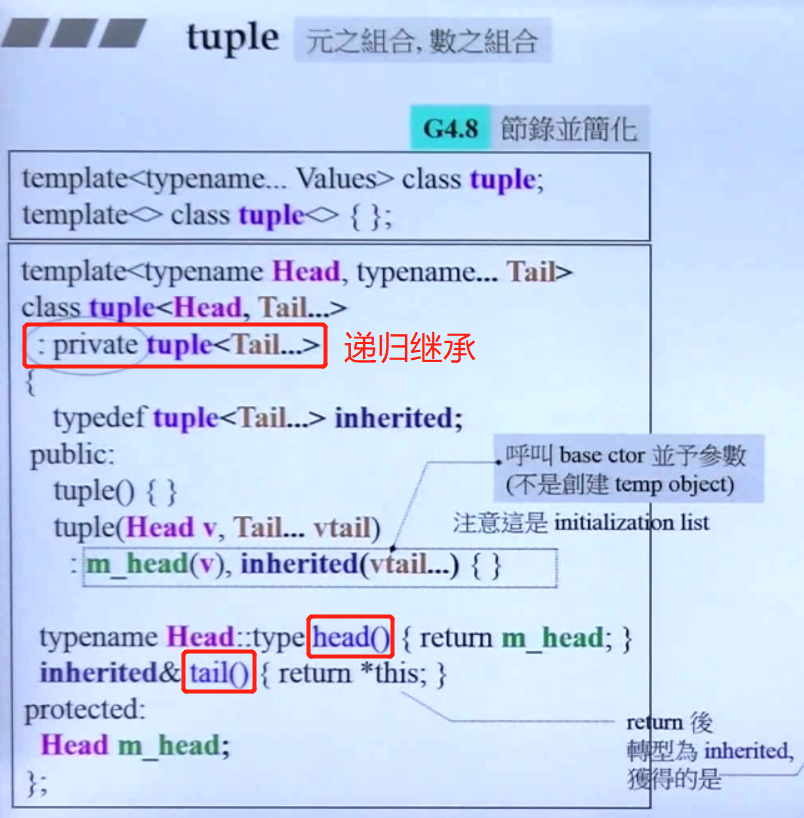
## (2)仿函数适配器——bind



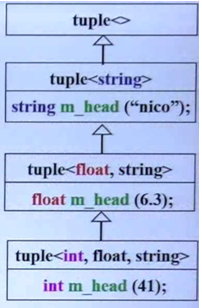
## (3)迭代器适配器



# 面试题型10：Tuple



**Tuple实现的关键：递归继承，见下图**



# 面试题型11：STL内存分配

## STL内存分配  <http://www.cnblogs.com/lang5230/p/5556611.html>

## STL 相关: STL 中的内存管理(allocator)的原理, 以及如何让它线程安全。  <http://yaocoder.blog.51cto.com/2668309/1208465/>