**STL源码剖析\_快学笔记**

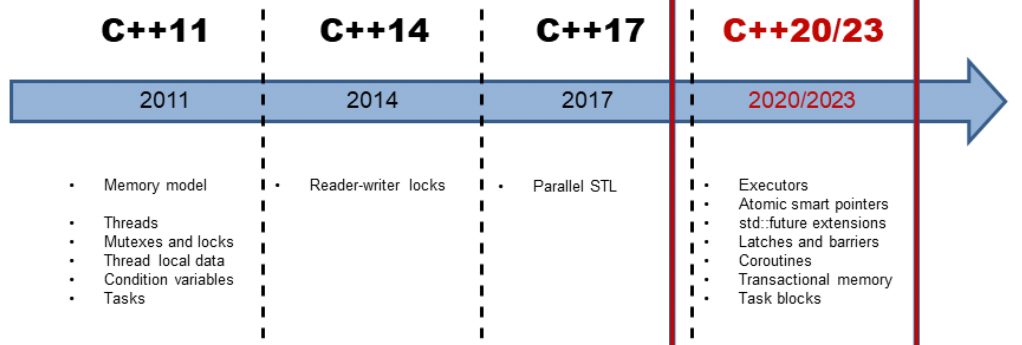
**前置知识**

**关于POD/Trival类型/not-Trival类型，**[**参考链接**](https://www.zhihu.com/question/472942396)

|  |
| --- |
| 这个概念有点复杂，我没搞懂。pod大概是一个和c兼容的类型，trival大概是说一个类的6类构造和析构函数写的很简单可以用memcopy，non-trival就是写的不简单因而不能用memcopy。   * trival dtor p108页 |

[并行和并发的区别？](https://paul.pub/cpp-concurrency/)系统支持处理多个任务，就是并发；如果操作系统有多核，也即能支持同一物理时间上多个任务同时运行，则为并行。并行是并发的子集，区别在与是否多处理器。

**C++14/C++17并发机制的更新？**



**1章 STL概论与版本简介**

1. STL是79年诞生，98年加入cpp标准库。C11出来之前boost库是常见的智能指针库，c11有自己的智能指针。且有多个STL版本，如SGI STL，GNU STL。
2. p19-p34，用判断各种宏定义是否存在，来define一些变量，就是可以在不同的平台上，有不同的源码定义的意思。

**2章 空间配置器**

* STL标准是要定义std::allocator，但SGI里面常用的确是std::alloc配置器p47。
* SGI STL具有次级SGI空间配置器的能力，比如vector<int, std::alloc> iv写法。
* SGI定义了一个simple\_alloc类【用alloc实现】，用于实现vector等等数据结构。

**STL 主要的内存配置器**

SGI定义了一个simple\_alloc类【用std::alloc实现】，用于实现两层配置器。p54、p55图展现关系、

* 主配置器：在申请大内存时使用。底层用malloc和free实现。set\_new\_handller()函数可以定义一个在oom时的
* 次配置器：如果申请小于128的内存则用此配置器。通过维护1个free-list，list含有16个块，每个块是一个链表，链表中还有块时就用，链表中块不够时去内存池memory pool取出一个块，供调用者使用。

|  |  |
| --- | --- |
|  |  |

**内存池p66：就是free-list中chunk\_alloc的实现。**[**参考知乎实现**](https://zhuanlan.zhihu.com/p/139835423)

chunk\_alloc(size\_t blockSize, int& requestNumberOfBlock)

* 内存池能满足要求 requestNumberOfBlock \* blockSize，就返回
* 内存池能够满足一个 requestNumberOfBlock，就返回1个快
* 1个内存快都不能满足：首先先将内存池中残余内存放入free\_list合适的块中，在向heap申请内存补充内存池。

**内存基本处理函数**

以下3个类型，对**pod类型数据**会执行较快的拷贝如（copy/fill/fill\_n)，非pod类型执行for遍历一个一个拷贝。p76

* uninitialized\_copy：对初始化的区域，用拷贝构造来生成对象
* uininitialized\_fill
* uininitialized\_fill\_n：对[Ft, Ft+n)

**3章 迭代器概念与traits编程技法**

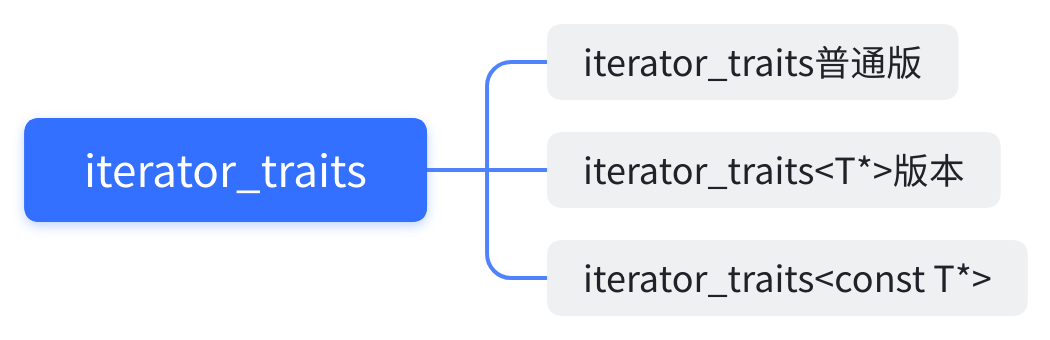
**STL设计容器&算法的思想**

1. 容器设计者：不仅要设计容器，还要尽可能基于iterator设计容器的5个typedef p84/89。
2. deque\_iterator虽然没有显示继承iterator实现，但它也实现了5个typedefp140。
3. 算法设计者：设计对不同读写能力迭代器的算法。STL算法的命名以其能接受的最低层次的迭代器命名，如advance(InputIterator& i, Distance n)可以接受Input、Forward、Bidirectional、Random Access四种类型的参数。
4. 总共是有input、ouput、forward、bidirectional、randomAccessIterator
5. 支持随机读写的容器：vector，deque，
6. 不支持随机读写的容器：list（底层链表）、pair（就两个元素）、map和set（红黑树或哈希表）

|  |
| --- |
| C++ template<class Category, class T, class Distance=ptrdiff\_t,   class Pointer=T\*, class Reference=T&> struct iterator{ // 相当于java的interface。  typedef Category iteator\_category;  typedef T value\_type;  typedef Distance difference\_type;  typedef Pointer pointer;  typedef Reference reference; }  // 某一个实现类应该是, 参考p100 template<class Item> struct ListIter: public std::iterator<std::forward\_iterator\_tag, Item>{} |

**Traits获取变量类型**

* iterator\_traits：负责萃取迭代器的各种类型
* \_\_type\_traits<T>：负责萃取pod相关的类型，p104

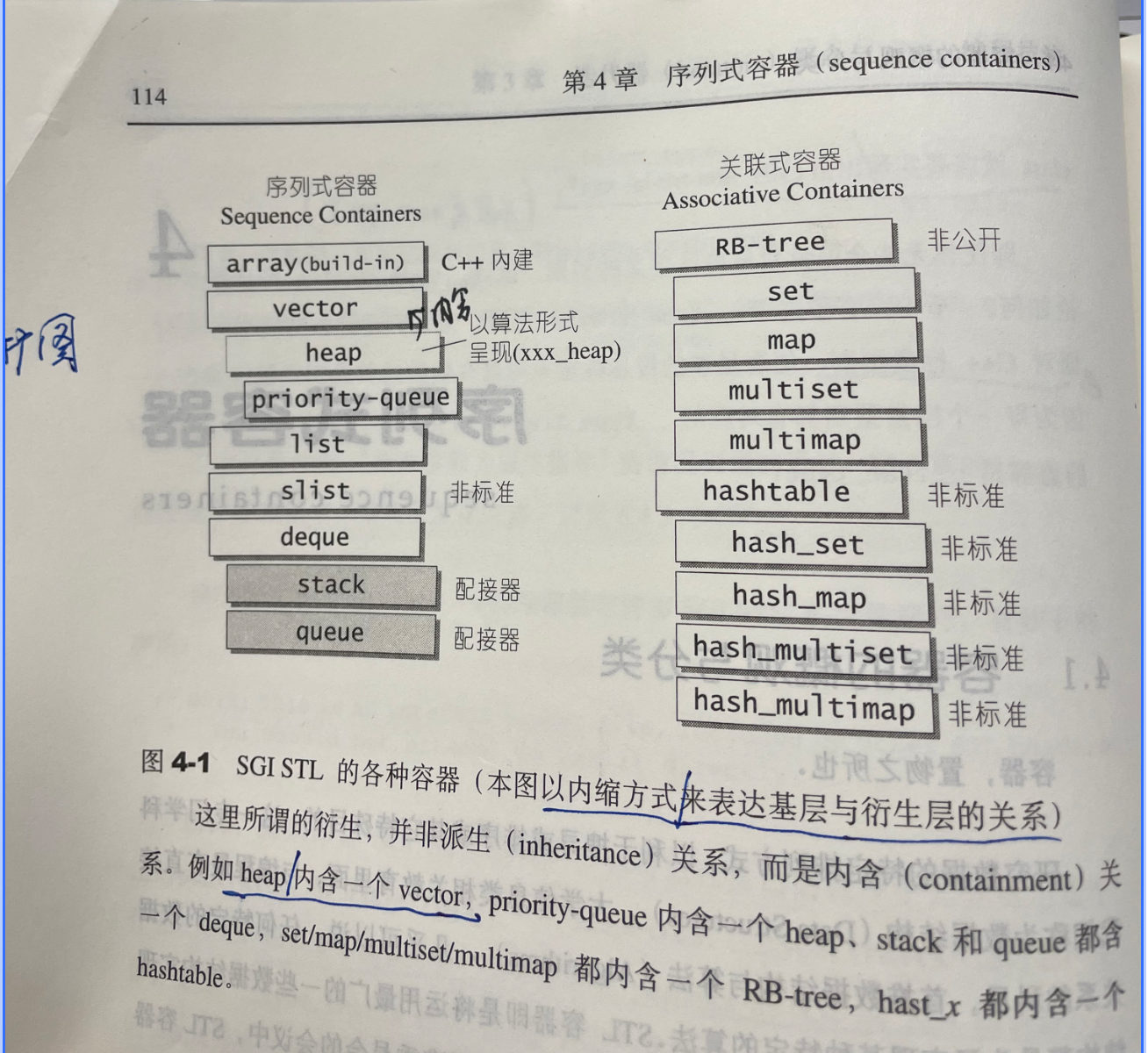


|  |
| --- |
| C++ template <class I> struct iterator\_traits{  typedef typename I::iterator\_category iterator\_category; // 帮助容器选择算法的 // func( iterator\_traits<INPUT>::iterator\_category()); // p99   typedef typename I::value\_type value\_type;  typedef typename I::difference\_type difference\_type;  typedef typename I::pointer pointer;  typedef typename I::reference reference; } |

如果没有反射，那怎么获取某个迭代器的类型呢？特性萃取iterator\_traits。怎么获取某对象的类型呢？type\_traits。

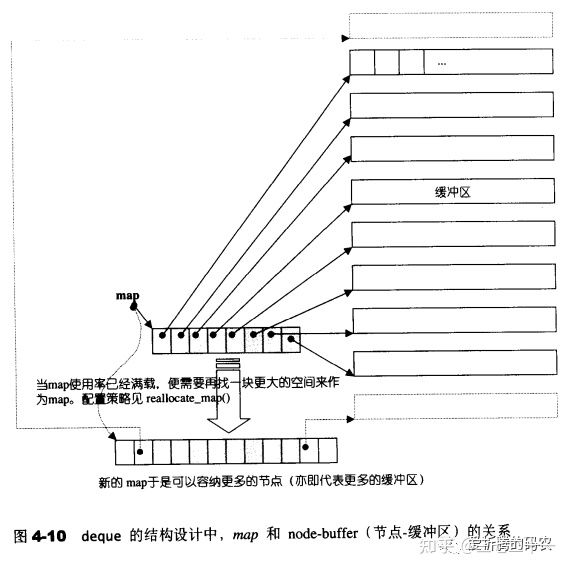
|  |
| --- |
| C++ template<typename \_Tp, typename \_Alloc = std::allocator<\_Tp> > class vector : protected \_Vector\_base<\_Tp, \_Alloc> { // 有删减  typedef \_Tp value\_type;  typedef typename \_Base::pointer pointer;  typedef typename \_Alloc\_traits::reference reference;  typedef \_\_gnu\_cxx::\_\_normal\_iterator<pointer, vector> iterator;  typedef size\_t size\_type;  typedef ptrdiff\_t difference\_type; } |

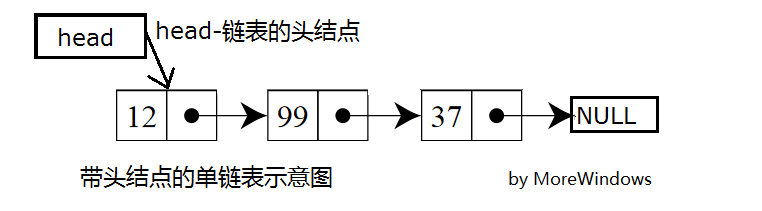
**4章 序列式容器**



|  |  |
| --- | --- |
| **容器结构** | **结构说明说明**  反正大家都是用simpe\_alloc分配和释放的，具体是啥得看结构。insert()、insert\_aux(iter, x)、erase、operatoer[]、 |
| vector | 1. 本质：用simple\_alloc分配和释放的，可~~变长，是连续的空间，~~ 2. ~~三个指针start、finish、end\_of\_stroge~~（最大水位）；如果水位满了，会resize()；   heap：用vector实现的堆，默认  priority\_queue：用vector【可以调整的】实现的优先级队列 |
| list | list本质：带有哨兵的双向链表 |
| deque | deque（双端队列）本质：利用复杂的map中控结构来管理，还可以随机读写的一个容器。   * 双向开口的，所以头尾插入效率较高 * 可随机读写的，random access * 关于deque的排序先赋值给vector，在用vector排序，这样效率高p143； * stack本质：是用deque/list管理的先入后出的栈 * queue本质：用deque/list管理的先入先出队列 |
| slist | 本质：是一个**只能头部插入（头插法的）单向的链表**   1. **slist和list底层都是链表，其迭代器不是通过计算偏移量得到的故而不失效**。比如list的insert， erase,splice，p186。slist一直头插法，insert、splice当然不失效； 2. **迭代器失效的情况：比如vector的insert会失效的。** |

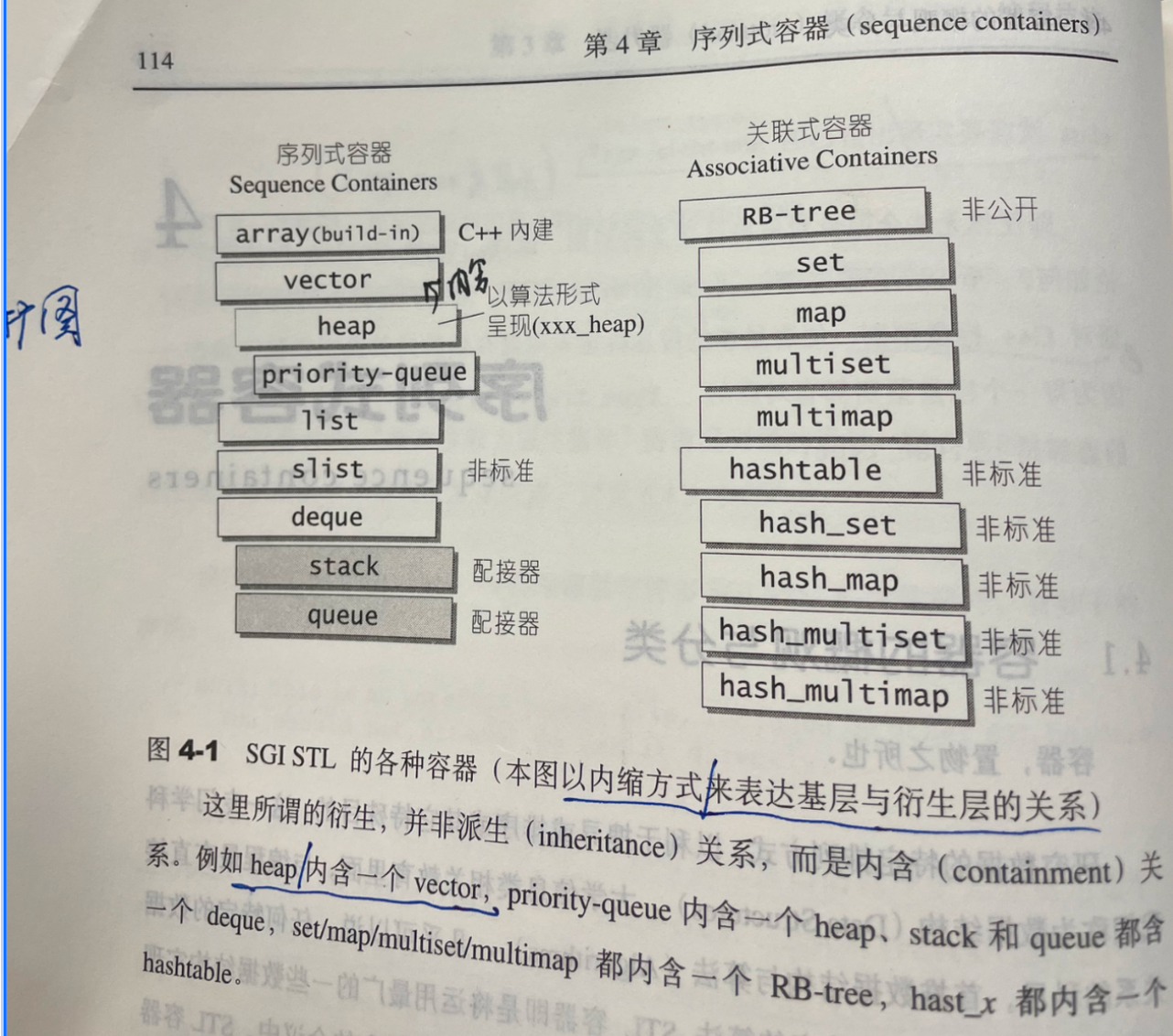






**5章 关联式容器**

就是形如key:value的容器，比如map和set，通常以rb-tree为底层的，或者以hashtable为底层。



**二分搜索树：**特例有平衡的二分搜索树

* 它的特例是AVL，相差为1
* 红黑树，

|  |  |  |
| --- | --- | --- |
| #include <map> | 1. 以红黑树RB-tree为底层的容器，就是以node(key+value）为节点构建的树。插入就会增加节点，删除就会减少节点，有插入删除就会有RB-Tree的旋转过程。  * 如果允许重复键值的插入：就有multiset、multimap； * 如果不允许重复键值的插入：就是map、set两个而已  |  | | --- | | p198-246，就总结出这么点点的东西。 | |
| #include <unordered\_map> | 1. 以hashtable为底层的容器，就是以桶和链表组成的表嘛。使用它就会有一些注意。 2. 如果一个插入操作让hashtable里元素个数>桶的个数，就会发生resize。桶的个数取值是比较固定的比如53、97、193等p257页有，每次取最接近的，且大于table元素个数的那个数。 3. 重要方法：insert\_unique()方法插入唯一值，insert\_equal方法插入可重复值。 4. hash\_function：很多对象不能hash，要自定义hash。 5. 自定义的类要自己写hash\_func 6. EqualKey = eqaul\_to<Key>： 7. 常见种类：hash\_set、hash\_map、hash\_multiset、hash\_multimap。 |

|  |
| --- |
| C++ // #include <unordered\_map> // GCC的源码  template <class Key, class T, class Hash = hash<Key>, class Pred = equal\_to<Key>,  class Alloc = allocator<pair<const Key, T>>> class unordered\_map { public:  // types  typedef Key key\_type;  typedef T mapped\_type;  typedef Hash hasher;  typedef Pred key\_equal;  typedef Alloc allocator\_type;  typedef pair<const key\_type, mapped\_type> value\_type;  typedef value\_type& reference;  typedef const value\_type& const\_reference;  typedef typename allocator\_traits<allocator\_type>::pointer pointer;  typedef typename allocator\_traits<allocator\_type>::const\_pointer const\_pointer;  typedef typename allocator\_traits<allocator\_type>::size\_type size\_type;  typedef typename allocator\_traits<allocator\_type>::difference\_type difference\_type;   } |

**6章 算法**

|  |  |
| --- | --- |
| **内容** | **算法名称** |
| **质变算法**   * inplace版本就地改元素 * copy版本是赋值到新位置改变，不改变元素 | * iter\_swap(IterLeft, IterRight) * replace(IterStart, IterEnd, old\_value, new\_value) * repalce\_copy(IterStart, IterEnd, IterResult, old\_value, new\_value) * replace\_if(IterStart, IterEnd, IterResult, old\_value, new\_value) * replace\_copy\_if(IterStart, IterEnd, IterResult, pre\_op, new\_value) * reverse(IterStart, IterEnd) * reverse\_copy(IterStart, IterEnd, IterResult) * rotate(IterStart, IterMiddle, IterEnd) * rotate\_copy(IterStart, IterMiddle, IterEnd, IterResult) * unique(IterStart, IterEnd) * unique\_copy(IterStart, IterEnd, IterResult) |
| **缺省/用户定义的函数**   * 缺省版本 * 用户自定义仿函数版本 | * accumulate(IterStart, iterEnd, init, binary\_op) * partial\_sum(IterStart, IterEnd, result, binary\_op) * fill(IterStart, IterEnd, value) * fill\_n(IterStart, nNumber, value) * lexicographical\_compare(IterStart1, IterEnd1, IterStart2, IterEnd2, comp) * mismatch(IterStart1, IterEnd1, IterStart2, binary\_pred) * copy(IterStart, IterEnd, IterResult) * copy\_backward(IterStart, IterEnd, IterResult) // 从右往左copy * adjacent\_find(ForwardIteratorStart, ForwardIteratorEnd, binary\_pred) |
| **双迭代器的函数** | * set\_union(IterStart1, IterEnd1, IterStart2, IterEnd2, result) * set\_intersection(IterStart1, IterEnd1, IterStart2, IterEnd2, result) * set\_difference(IterStart1, IterEnd1, IterStart2, IterEnd2, result) * set\_symmetric\_difference(IterStart1, IterEnd1, IterStart2, IterEnd2, result) |
| **其他算法** | * count(IterStart, IterEnd, value) * count\_if(IterStart, IterEnd, pred\_op) * find(IterStart, IterEnd, value) * find\_if(IterStart, IterEnd, pred\_op) * find\_end(IterStart, IterEnd, value)，内部根据Iter的前向、双向能力由4个子函数组成 * for\_each(IterStart, IterEnd, function\_op) * generate(IterStart, IterEnd, gen\_op) // 根据gen\_op来生成元素 * generate\_n(IterStart, n, gen\_op) * includes(IterStart1, IterEnd1, IterStart2, IterEnd2, comp\_op) // Ite1包含Iter2吗 * max\_element(IterStart, IterEnd, comp\_define) * remove(IterStart, IterEnd, value) // 移除元素value，但是最后不会删除 * remove\_copy(IterStart, IterEnd, result, value) * lower\_bound(IterStart, IterEnd, value) // 应用于有序区间 * upper\_bound(IterStart, IterEnd, value) // 应用于有序区间 * pair<> equal\_range(IterFirst, IterEnd, value) |
| **内置算法** | * binary\_search(IterFirst, IterEnd, value) * binary\_search(IterFirst, IterEnd, value, comp\_op) * next\_permutaion(IterFirst, IterEnd) * prev\_permutaion(IterFirst, IterEnd) * random\_shuffle(IterFirst, IterEnd) * partial\_sort(IterFirst, IterMiddle, IterEnd)，使得[First, Middle)是[First, end)中有序，其他元素放在后面。 * partial\_sort\_copy(IterFirst, IterEnd, IterLast) * sort(IterFirst, IterEnd) // 数据量小用的插入排序，数据量大用的快速排序（特殊的快排median-of-three）。 * inplace\_merge(IterFirst, IterMiddle, IterResult） // 原地排序算法 * nth\_element(IterFirst, IterNth, IterEnd) // 调整后nthIter位置是第Nth个大小的值，其他值不确定 * merge\_sort(IterFirst, IterEnd) // 归并排序 |

**remove(IterFirst, IterEnd, value）需要注意的点**

|  |
| --- |
| Plain Text 6/8/10/3/4/5/6/6/6/7/8 // 如果remove(first, last, value) 8/10/3/4/5/7/8[6/6/7/8] |

**7章 仿函数**

定义：一个看起来像函数一样可以调用的类，本质上该类实现了operator()

常用于：STL算法中的某个参数，比如FuncOp，PreOp，UnaryOp，BinaryOp

* unary\_function struct，定义了一元函数的参数型别，和返回值的型别。p416
* binary\_funciton struct，定义了二元函数的参数型别，和返回值的型别。

|  |
| --- |
| C++ template <class Arg, class Result> struct unary\_function {  typedef Arg argument\_type;  typedef Result result\_type; };  template<class T> struct negate: public unary\_function<T,T> {  T operator()(const T& x) const { return -x; } }; // auto ne\_10 = negate()(10); // ne\_10 = -10;  template<class Predicate> class unary\_negate { public:  bool operator()(const typename Predicate::argument\_type& x) const {  // ....  } }; |

**仿函数的种类**

|  |  |
| --- | --- |
| **仿函数的种类** | **函数名称** |
| **算术类** | * plus<T>， // auto res = plus<T>()(1, 2); * minus<T> * mutiplies<T> * divides<T> * modules<T> * negate<T> |
| **关系运算类** | * equal\_to<T> * not\_equa~~l~~\_to<T> * greater<T> * greater\_equal<T> * less<T> * less\_equal<T> |
| **逻辑运算类** | * logical\_and<T>, // logail\_and<int>()(true, true); * logical\_or<T> * logical\_not<T> |
| **正同、选择、投射**  **p423** | * identity<T,T>， // 正同 * select1st<> * select2nd<> * select1st<T1, T2> // 应用于 pair * select2st<T1, T2> // 应用于 pair |

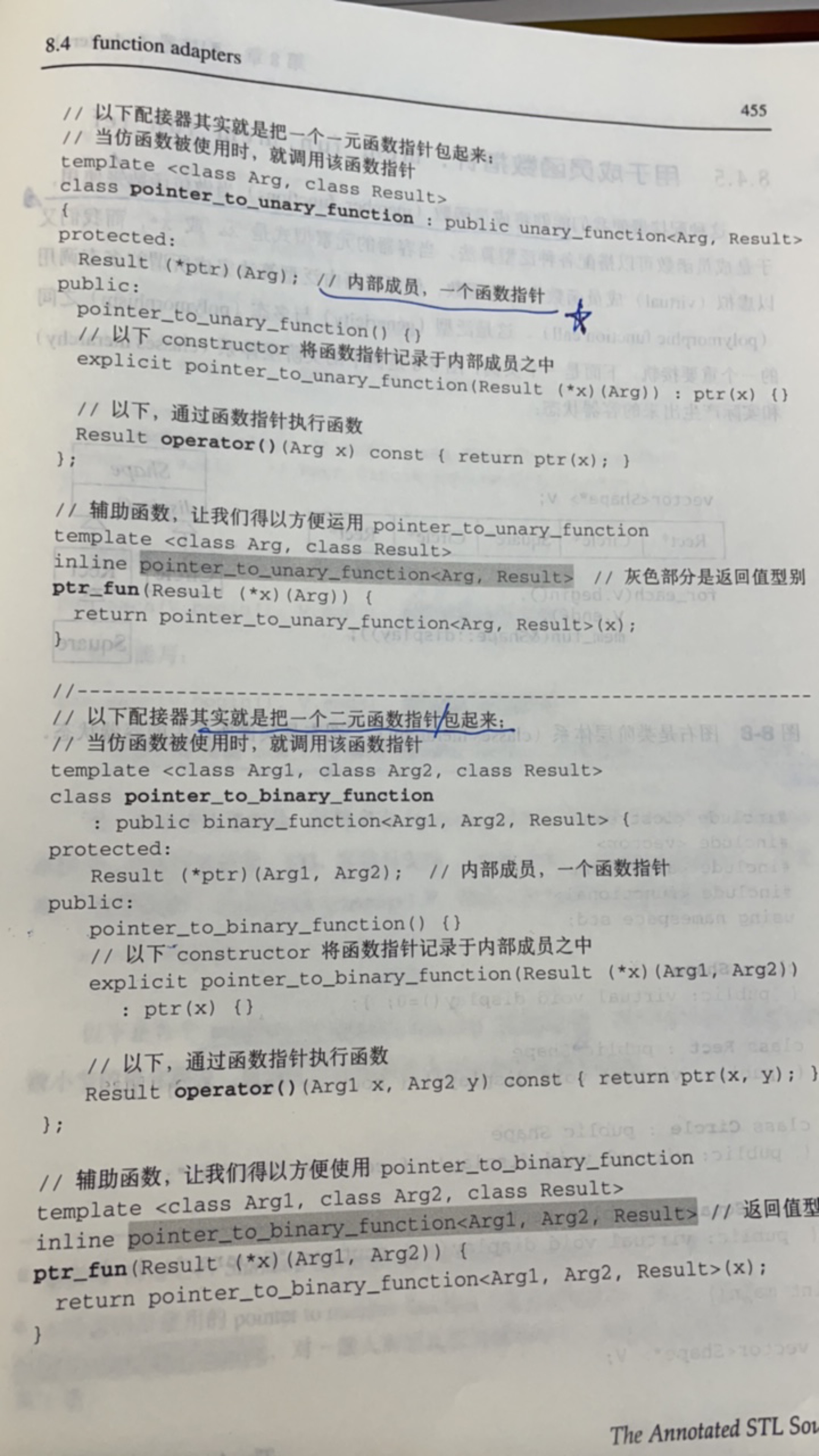
**8章 配接器 Adapters**

**配接器Adapters 的分类**

* 改变容器的接口，就是给容器套个壳子（有一个Struct的member是容器)，ContainerAdapter
* 改变迭代器的接口，就是给迭代器套个壳子（有一个Struct的member是迭代器），IteratorAdapter
* 改变函数的接口，就是给函数套个壳子(有一个Struct的member是仿函数)，FuncitonAdapter

|  |  |
| --- | --- |
| **配接器种类** | **举例子** |
| **ContainerAdapter**   * 内藏了一个 container object | 内涵一个容器，包装该容器的能力就OK了   * queue，先入先出 * stack，先入后出  |  | | --- | | C++ // p434 template<class T, class Sequence = deque<T> > // 还可以用list class stack { protected:  Sequence c; // 底层容器 }; | |
| **IteratorAdapter**   * 内藏了一个 iterator object | * insert\_iterators：将对iter的赋值assgin操作，转换为对容器的insert操作。 * back\_insert\_iterator，p435，其内部包含一个容器，push\_back()来给内涵 * front\_insert\_iterator，注意因为vector, slist没有push\_front()，所以不能应用front\_insert\_iterator * insert\_iterator： * reverse\_iterators：如果stack、queue，priority\_queue不提供begin(), end()所以没有单向迭代器，书中p438 * iostream\_iterators：p442  |  | | --- | | C++ template<class Container> class back\_insert\_iterator{ protected:  Container\* container; public:  operator=(const typename Container::value\_type& value) {  container->push\_back(value);  return \*this;  } }  template<class Container> inline back\_insert\_iterator<Container> back\_inserter(Container& x) {  return back\_insert\_iterator<Container>(x); } // copy(pa\_vec.cbegin(), pa\_vec.cend(), back\_inserter(re3)); // -1 0 1 2 3 4 5  istream\_iteraor<int> inite(cin), eos; copy(inite, eos, inserter(id, id.begin()); | |
| **FucntionAdapters**   * 内藏了一个 member object | * not1()是一个辅助函数，它实际上会产生unary\_negate<Pred>(pred) * ptr\_fun()是一个辅助函数，它实际上用ptr\_fun可以把一个一元、二元的函数指针包起来，就可以给STL用了。 * mem\_fun()、mem\_fun\_ref()：用于成员函数的仿函数指针， * ...  |  | | --- | | C++ template<class Predicate> class unary\_negate: public unary\_function<typename Predicate::argument\_type, bool> { protected:  Predicate pred; // 可调对象，仿函数 public:  bool operator() (const typename Predicate::argument\_type& x) const{  return !pred(x);  } }  template <class Predicate> inline unary\_negate<Predicate> not1(const Predicate& pred) {  return unary\_negate<Predicate>(pred); } | |

**ptr\_fun实现**



**mem\_fun、mem\_fun\_ref实现**

|  |  |  |
| --- | --- | --- |
|  |  |  |

**STL算法中的unary\_op，binary\_op位置通常可以放置如下的几种内容p432:**

* 函数指针，比如 for\_each(vec.begin(), vec.end(), print)
* 仿函数。 ptr\_fun(print)，自己定义。
* 修饰后的类成员函数，比如 mem\_fun，mem\_fun\_ref 去包装一下类里面的成员指针

**---彩蛋-------------------------**

1. **同行可添加微信，专搞CPP和基架**

|  |
| --- |
| 大话孙，base北京or杭州，22毕业北邮，四川，邓小平&朱德老家离我家不到100公里，最重要的是，那里不还有我吗。  up主在22-24年内视频，主要面向0-3年内的工作经验的cpp相关研发分享，大佬们要多提意见别噎着不说话，同行们也请多多支持。哈哈哈  对了，虽然我琴棋书画无一精通，但架不住我总在参与呀。所以以后类似跑步、游泳、旅游等活动都可以叫我，我这毕竟上得厅堂下得厨房，全菜但我全会。同时新晋为萌新up主“大话孙”，先录一些技术书的学习视频、程序员业务生活等，再录啥没想好，求一键三连和关注，哪怕提点意见也行。  最后，在B站台分享和记录生活吧，期待在B站交到好朋友。总之，工作了就要快快乐乐把工作办了，想必一定是精彩的旅程。  人生啊，妙不可言。up up up ! |



2. [**下一期录什么，你的投票来定**](https://qppw4bc6rk.feishu.cn/wiki/wikcneCncjQ8eDI07UooY8FT8Db)

|  |
| --- |
| 感谢路过的朋友，兄弟姐妹们的支持。对了下一个视频录什么，可以投票了，先谢谢你的支持了。 |

3. **ANRR，懂自懂**