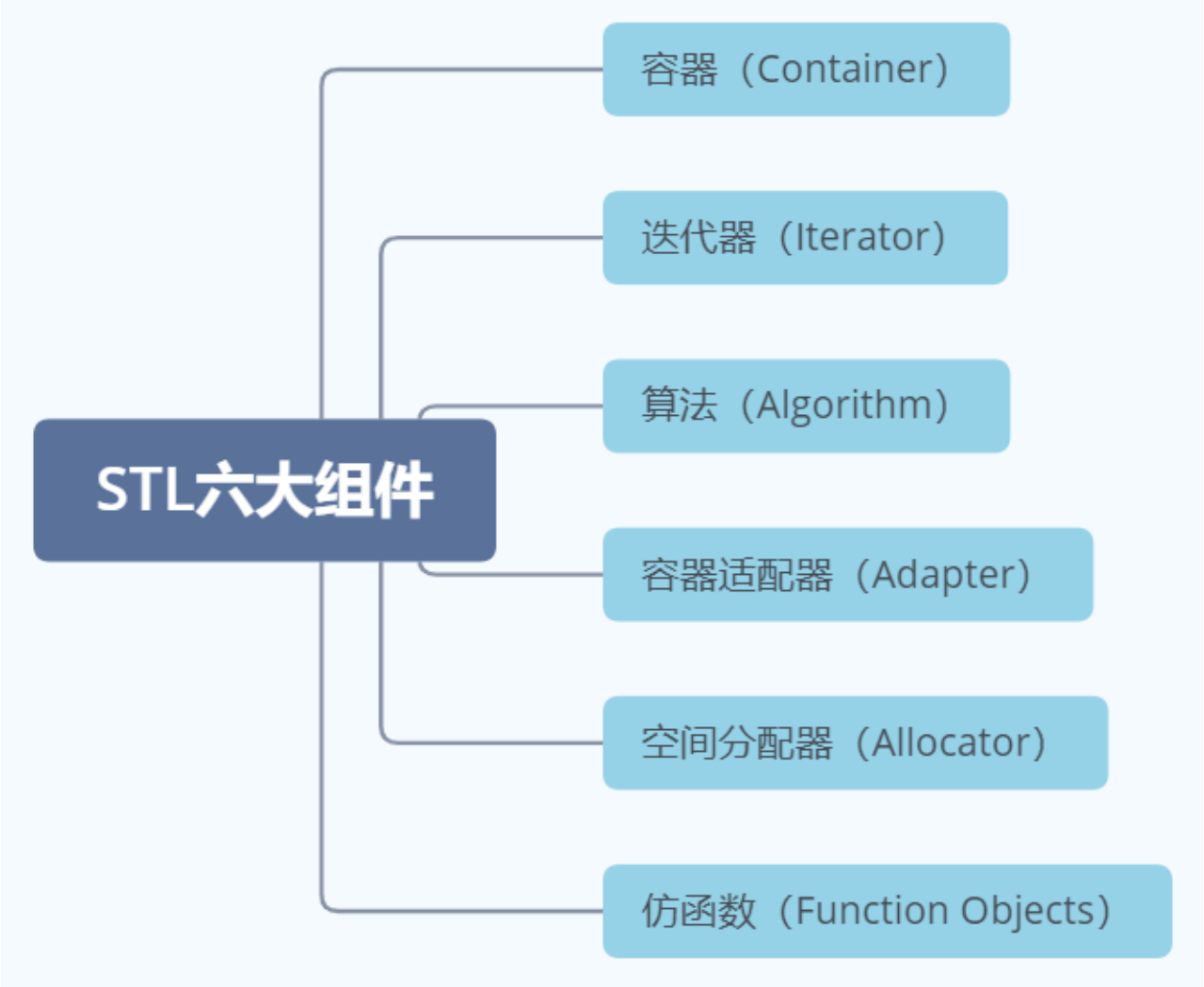
第12章-STL

author：岳石磊 copyright：科林明伦 内部资料禁止外泄

1. 简述

STL 是“Standard Template Library”的缩写，中文译为“标准模板库”。STL 是 C++ 标准库的一部分，位于各个 C++ 的头文件中，即它并非以二进制代码的形式提供，而是以源代码的形式提供。STL体现了泛型编程的思想，大部分基本算法被抽象，被泛化，独立于与之对应的数据结构，用于以相同或相近的方式处理各种不同情形，为我们提供了一个可扩展的应用框架，高度体现了程序的可复用性。STL的一个重要特点是数据结构和算法的分离。
STL的头文件都不加扩展名，且打开std命名空间。
包含了六大组件：



2. 容器

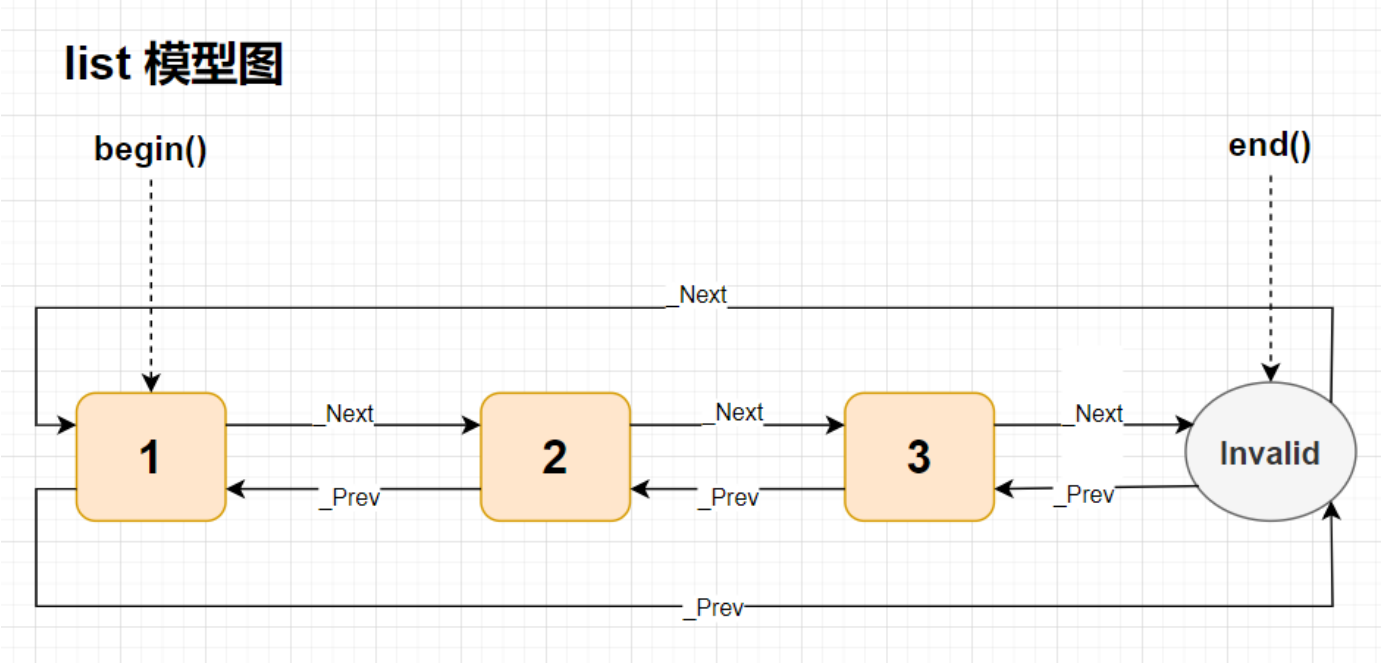
主要分两大类：
序列性容器：序列容器保持插入元素的原始顺序。允许指定在容器中插入元素的位置。每个元素都有固定位置，取决于插入时机和地点和元素值无关，如：链表（list），向量（vector），双端队列

（deque）。
关联性容器：元素位置取决于特定的排序规则和插入顺序无关，map、hash-map、set。
容器类自动申请和释放内存，无需new和delete操作。

2.1 list链表

STL链表是序列性容器的模板类，它将其元素保持在线性排列中，链式结构，并允许在序列中的任何位置进行有效的插入和删除。
特点：list在任何指定位置动态的添加删除效率不变，时间复杂度为`O(1)`，操作相比于vector比较方便。但其查找效率为`O(n)`，若想访问、查看、读取数据使用vector。

双向循环链表：



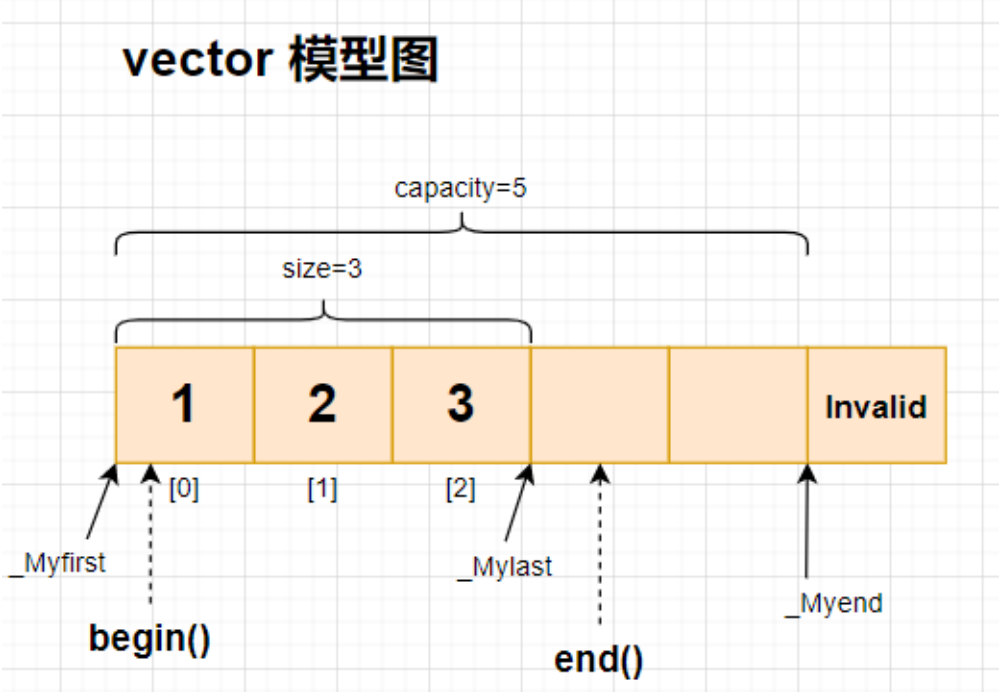
require：#include< list > using namespace std;
list(size_type Count)。list(size_type Count,const Type& Val)。构造链表长度并设置初始值。有默认值。

1. begin()：获取头结点的迭代器。end()：获取尾节点的迭代器。
2. front()：返回头结点里的值。 back()：返回尾结点里的值。
3. clear()：清空链表。
4. size(); 返回链表的长度、元素的数量。
5. bool empty()，链表是否为空。
6. erase()：删除指定迭代器位置的节点，返回的是删除节点的下一个节点的迭代器。 insert()：指定迭代器位置插入一个元素，返回的是插入的元素的迭代器。
7. push_back()、push_front()、pop_back()、pop_front()：链表头尾添加、删除。
8. remove(const Type& _Val)：将值为val的所有节点删除。
9. unique()：将连续而相同的节点移除只剩一个。
10. sort()：对链表元素进行排序，默认升序。如果要指定排序规则，需要指定排序规则函数。bool func(Type,Type); 或 greater() 降序，less() 升序。
11. reverse()：链表进行翻转。

12. splice(iterator Where, list& Right)：将 Right 链表整个结合到另一个链表where位置之前，这是一个“剪切”操作，Right 链表将为空。
    splice(iterator Where,list& Right,iterator First)：将Right链表的First位置节点结合到 this 链表的where位置之前，这是一个“剪切”操作。this 和 Right 可以为同一个链表。
    splice(iterator Where,list& Right,iterator First,iterator Last)：将Right链表的First位置到Last位置的一段元素[First,Last)，不包含Last，结合到this链表的where位置之前，这是一个“剪切”操作。this 和 Right 可以为同一个链表，但 Where不能位于[First,Last) 内。
13. merge(list& Right, Traits Comp); 将 Right 链表合并到this链表上， this 和 Right 必须经过排序，两者或都为递增、或都为递减，Comp 描述了递增合并还是递减合并，bool func(Type,Type); 或 greater() 降序，less() 升序。这是一个“剪切” 操作，Right 将为空链表。
14. swap(list& Right); 交换两个链表。

2.2 vector向量

vector 的行为类似于数组(array)，但是其容量会随着需求自动增长。向量在尾部的push和pop操作时间是恒定的。在向量的中间insert或erase元素需要线性时间，在序列结尾插入、删除操作比开始位置性能要优越。
当向量元素增加超过当前的存储容量时，会发生重新分配操作。重载了[] 操作符，就意味着它可以像数组一样使用【下标】访问元素。
特点：数据的存储访问比较方便，可以像数组一样使用`[index]`访问或修改值，适用于对元素修改和查看比较多的情况，对于insert 或 erase 比较多的操作，很影响效率，不建议使用vector。



require：#include< vector > using namespace std;
vector(size_type Count)。 vector(size_type Count,const Type& Val)。构造函数，构造指定长度的向量并设定初始值。有默认值。

1. begin()，end()，返回向量头尾元素的迭代器。
2. front()，back()，返回头尾元素中的值。
3. size()，返回向量的使用量，capacity()，返回向量的容量。

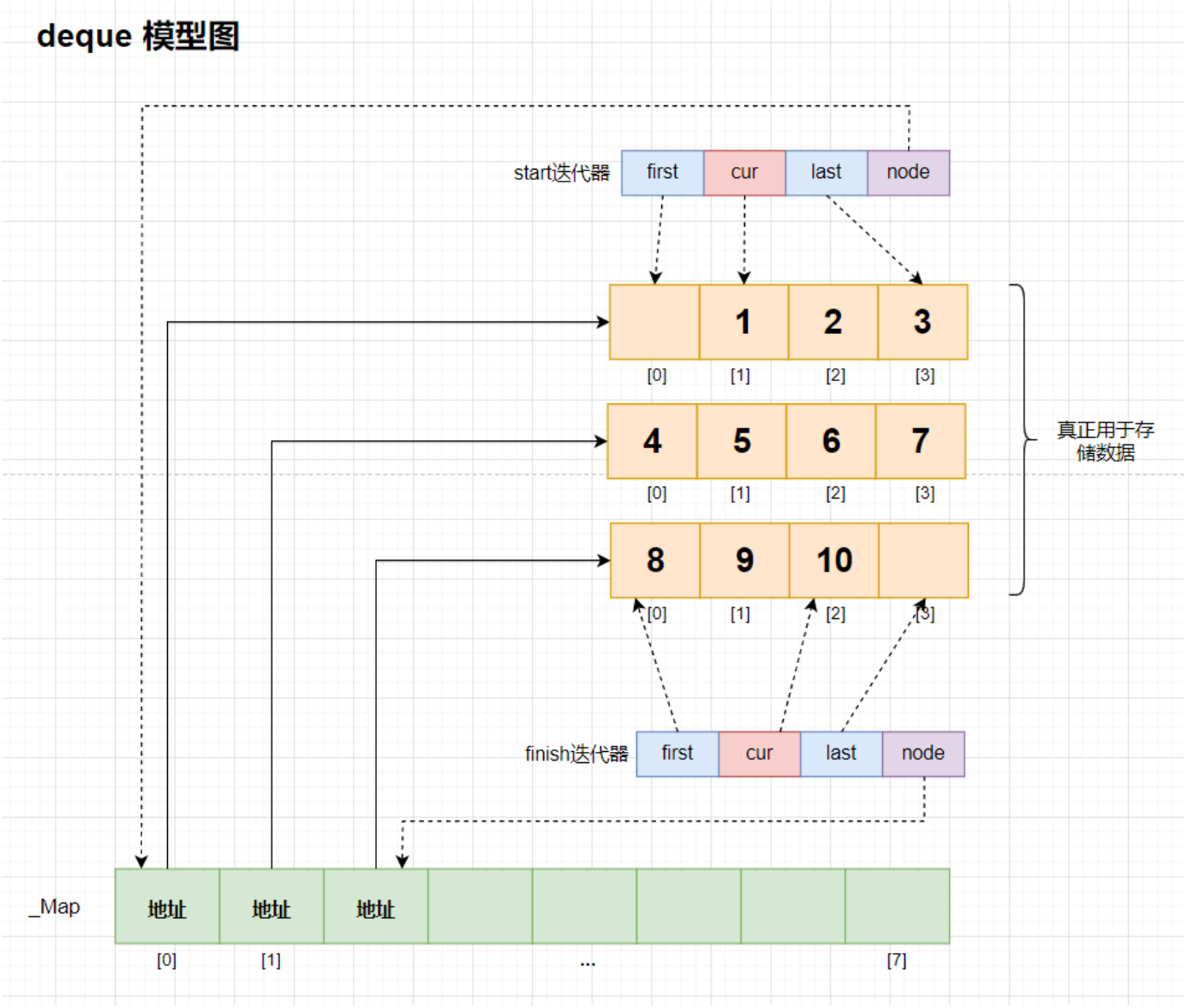
4. push_back()，pop_back()，在向量头、尾添加删除，当用push_back 向vector 尾部加元素的时候，如果当前的空间不足，会重新申请一块更大的空间。pop_back 删除时，使用量减少，但容量不会减少。不同于list，并没有提供 push_front 和 pop_front。
5. insert(const_iterator Where,const Type& Val)，向量的某个位置之前插入指定值。
   insert(const_iterator Where, size_type Count, const Type& Val)。向量的某个位置之前插入count个指定值。
   返回插入元素的迭代器，size会增加。
6. erase(const_iterator Where)，删除迭代器指向的元素，返回的是删除元素的下一个元素的迭代器。size 减少，capacity 不变。
7. clear()，清空向量元素，size 使用量为0，capacity 不变。
8. empty()，向量是否为空，描述的是使用量。
9. resize(size_type Newsize)，resize(size_type _Newsize,Type _Val)。为向量指定新的使用量，如果使用量减少，元素按顺序截取，但容量不变，如果使用量增加大于原容量，则扩展容量，并可以指定新扩展元素的值。
10. swap(vector& Right)，交换两个向量的元素。

对比 `list` 和 `vector`

1. vector 是连续性空间，顺序存储，list 链式结构体，链式存储。
2. vector在非尾部插入、删除节点会导致其他元素的拷贝移动，list则不会影响其他节点元素。
3. vector 一次性分配好内存，使用量不够时，申请内存重新分配。list每次插入新节点时都会申请内存。
4. vector随机访问性能好，插入删除性能差；list随机访问性能差，插入删除性能好。
5. vector 具有容量和使用量的概念，而list只有使用量（即长度）概念。

2.3 deque双端队列

deque 没有容量的观念。它是动态以分段连续空间组合而成，一旦有必要在deque 的前端和尾端增加新空间，串接在整个deque 的头端或尾端，deque 的迭代器不是普通的指针，其复杂度比vector 复杂的多。除非必要，我们应该尽量选择使用vector 而非 deque。
deque 是一种双向开口的连续性空间，可以在头尾两端分别做元素的插入和删除操作。

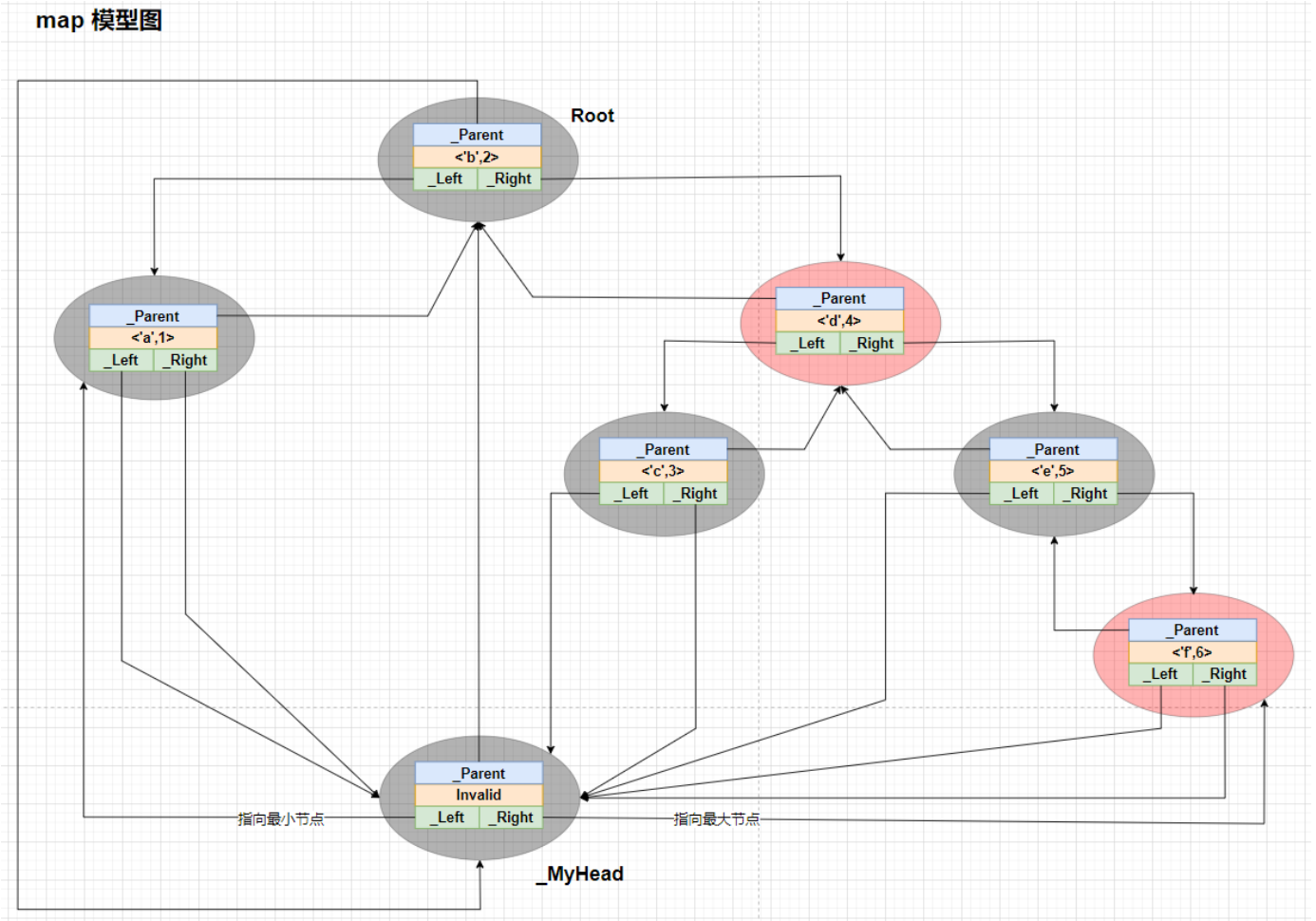


require：#include< deque > using namespace std;
deque(size_type Count)，deque(size_type _Count,const Type& Val)，构造函数，构造指定长度的向量并设定初始值。有默认值。

1. push_back()、push_front()、pop_back()、pop_front()。
2. begin()、end()、back()、front()、erase()、insert().
3. size()，empty()，clear()，支持[]下标访问。

2.4 map 映射表

Map 的特性是，所有元素都会根据元素的键值自动被排序，map的所有元素都是pair，同时拥有键值(key)和实值(value)。Pair的第一元素被视为键值，第二元素被视为实值。Map 不允许两个元素拥有相同的键值，map 的键值关系到map的元素的排列规则，任意改变map元素键值将严重破坏map的组织。所以不可以通过map 的迭代器来改变map 的键值。但是可以通过迭代器来修改元素的实值。
查找效率：O(log2(n))，内部实现红黑树。



require：#include< map > using namespace std;

1. begin()、end()， 支持迭代器遍历。
2. pair <iterator, bool> insert(value_type& Val)，插入一个元素，如果key值重复，则插入失败。iterator erase(iterator Where)，返回删除的下一个。
3. clear()、size()、empty()
4. iterator find(const Key& Key)、按键值查找，未匹配到返回end()，count(const Key& Key)，按键值统计元素，返回1 或 0。
5. upper_bound(const Key& Key)、返回大于该键值的map 的迭代器。lower_bound(const Key& Key)，返回该键值或者大于该键值的map 的迭代器.

2.5 set 集合

所有元素都会根据元素的键值自动被排序，Set 的元素不像Map那样可以同时拥有实值和键值，Set 元素的键值就是实值，实值就是键值。Set 不允许两个元素有相同的键值，因为Set 元素值就是其键值，关系到 Set 元素的排列规则。如果任意改变Set 的元素值，会严重的破坏Set组织。
查找效率：O(log2(n))，内部实现红黑树。
require：#include< set > using namespace std;

1. begin()、end()， 支持迭代器遍历。
2. pair<iterator, bool> insert(const value_type& Val)，插入一个元素，如果key值重复，则插入失败。iterator erase(iterator Where)，返回删除的下一个。
3. clear()、size()、empty()

4. iterator find(const Key& Key)、按键值查找，未匹配到返回end()，count(const Key& Key)，按键值统计元素，返回1 或 0。
5. upper_bound(const Key& Key)、返回大于该键值的set 的迭代器。lower_bound(const Key& Key)，返回该键值或者大于该键值的set 的迭代器.

2.6 hash_map哈希表

基于hash table（哈希表），数据的存储和查找效率非常高，几乎可以看作常量时间，相应的代价是消耗更多的内存。使用一个较大的数组来存储元素，经过算法，使得每个元素与数组下标有唯一的对应关系，查找时直接定位。
require：#include< hash_map > using namespace std;
若高版本需要 `#define _SILENCE_STDEXT_HASH_DEPRECATION_WARNINGS` 来去除error，或使用 `<unordered_map>`
查找效率：O(1)

1. begin()，end()，返回头、尾节点的迭代器。支持迭代器遍历。
2. pair<iterator, bool> insert(const value_type& Val)，插入一个元素，如果key值重复，则插入失败。iterator erase(iterator Where)，返回删除的下一个。
3. iterator find(const Key& _Key)，按照key值进行查找，
   注意：查找速度, 数据量, 内存使用

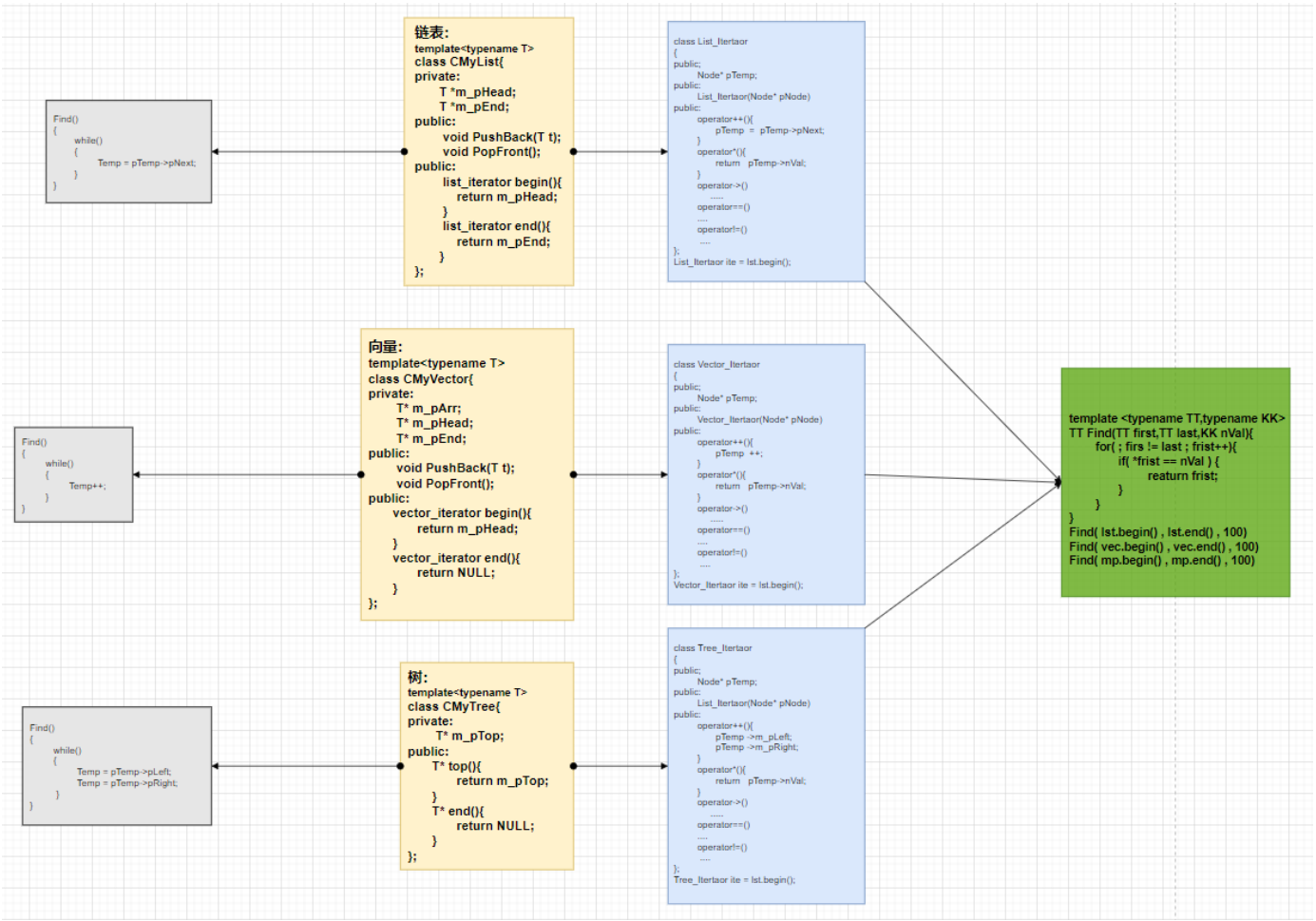
3. 算法

STL 中的算法头文件位于< algorithm > 和 < numeric > 文件中（以 < algorithm > 为主）。

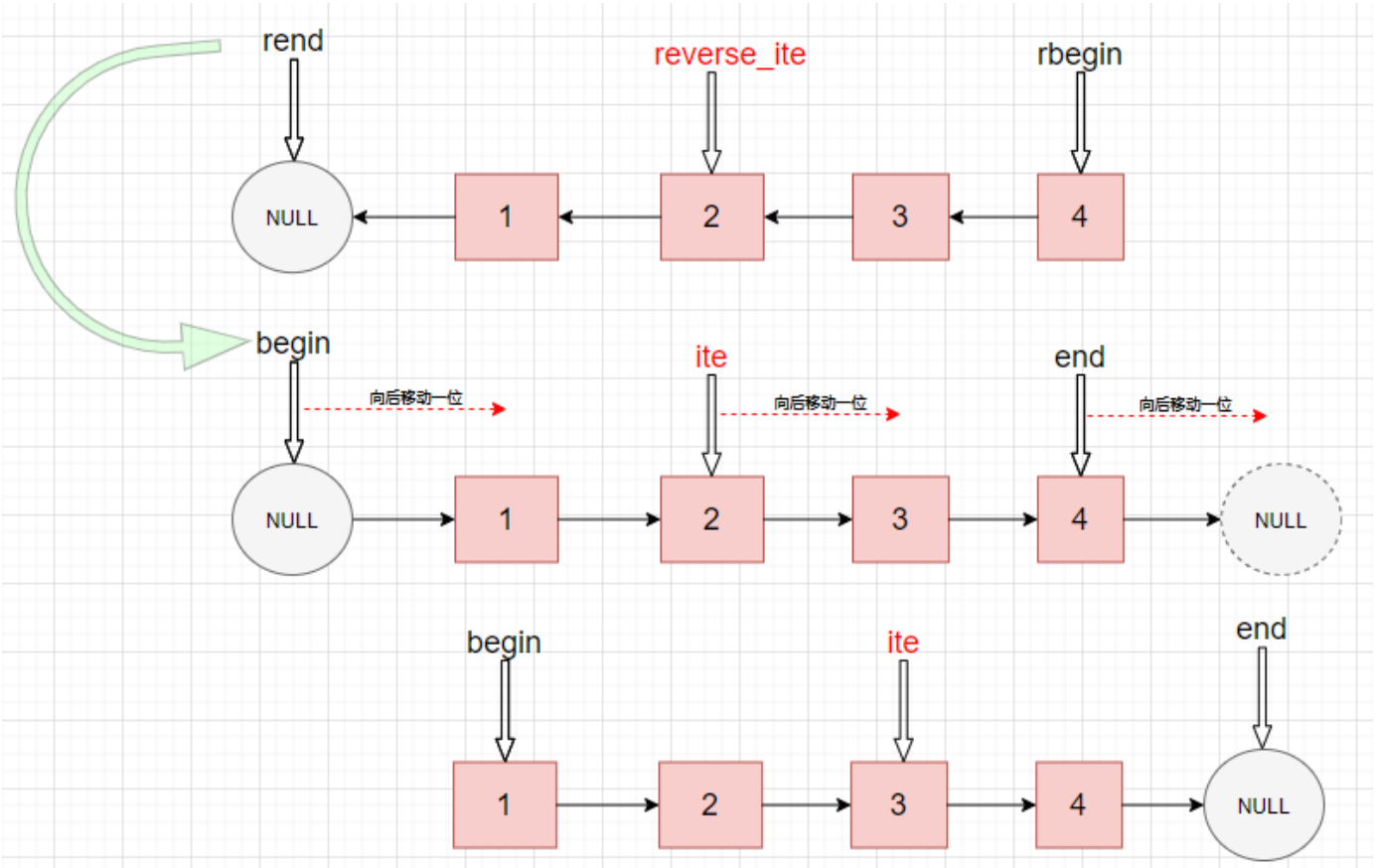
1. for_each(InputIterator First, InputIterator Last, Function _Func);
2. count(InputIterator First, InputIterator Last, const Type& Val)，统计指定范围内的元素的数量。
3. bool equal(InputIterator1 First1, InputIterator1 Last1, InputIterator2 First2)， 比较first1~last1 范围内的元素的值，与起始的First2 逐个比较是否相等。相等返回true，否则返回false。First2 容器元素数不能小于First1~Last1。
4. InputIterator find(InputIterator First, InputIterator Last, const Type& Val)。
5. void sort(RandomAccessIterator first, RandomAccessIterator last, Predicate comp); 不能对链表进行排序，链表自带了sort功能函数。
6. const Type& max(const Type& Left, const Type& Right); 返回相比最大的容器，是复制一份。
7. const Type& min(const Type& Left, const Type& Right); 返回相比最小的容器，是复制一份。

4. 迭代器

算法-迭代器-容器之间的关系。



反向迭代器： reverse_iterator，rbegin()，反向的头，rend()，反向的尾。
reverse_iterator::base()，反向迭代器转正向迭代器。



5.容器适配器

容器适配器即是对特定类封装作为其底层的容器，并提供一组特定的成员函数来访问其元素。容器适配器本质上还是容器，只不过此容器模板类的实现，利用了大量其它基础容器模板类中已经写好的成员函数。当然也可以添加新的成员，它是一个封装了序列性容器的模板类，它在一般序列容器的基础上提供了一些不同的功能。之所以称为适配器类，是因为它可以通过适配器现有的接口来提供不同的功能，将不适用的序列式容器（包括 vector、deque 和 list）变得适用。容器适配器不支持迭代器、当然也就不能使用算法（algorithm）函数。
注意：容器适配器默认都是用底层**序列性容器**实现的。
容器适配器主要包括：stack栈适配器、queue队列适配器。

5.1 栈（stack)

实现的是一个后入先出（Last-In-First_Out, LIFO）的压入栈。它默认是用 deque<_Ty>去实现的，但也可以用list vector等底层容器实现。

功能	含义
size	Returns the number of elements in the stack.
empty	Tests if the stack is empty.
pop	Removes the element from the top of the stack.
push	Adds an element to the top of the stack.
top	Returns a reference to an element at the top of the stack.

5.2 队列(queue)

特点：实现的是一个先入先出（First-In-First_Out, FIFO）的队列，它默认是用 deque<_Ty>去实现的，但也可以用list等底层容器实现。

功能	含义
empty	Tests if the queue is empty.
size	Returns the number of elements in the queue.
back	Returns a reference to the last and most recently added element at the back of the queue.
front	Returns a reference to the first element at the front of the queue.
pop	Removes an element from the front of the queue.
push	Adds an element to the back of the queue.

6. 仿函数

仿函数的通俗定义：仿函数（functor）又称为函数对象（function object）是一个能行使函数功能的类。仿函数的语法几乎和我们普通的函数调用一样，不过作为仿函数的类，都必须重载operator()运算符。