C++ STL 学习

```
https://www.fynote.com/d/2747
```

1. 基本概念

STL 是惠普 实验室开发的一系列软件的统称，主要出现在C++ 中，但在被引入C++ 之前该技术就已经存在了很长的一段时间

STL 从广义上分三类： algorithm (算法)， container (容器)， iterator(迭代器)，容器和算法通过迭代器可以进行无缝连接， 几乎所有的代码 都采用了模板类和模板函数的方式，这相比于传统的函数和类组成的库说 提供了更好的代码重用的机会

在C++ 标准中。STL 被组织为下面13个头文件，

```
<algorithm>  <deque> <functional> ,<iterator>,<vector> ,<list> <map> ,
<memory> <numeric> <queue> <set> <stack> <utility>
```

使用STL的好处

1） STL 是C++的一部分，因此不需要额外安装什么，它被内建在你的编译器之内

2） STL的一个重要特点是数据结构和算法的分离，尽管这是一个简单的概念，但是这种分离确实使得STL变得非常通用

3） 程序员可以不用考虑STL具体的实现过程，只需要熟练使用STL 就OK了，这样就可以把精力放在程序开发别的方面

4） STL 具有高可重用性，高性能，高移植性，跨平台的优点

- STL 是由容器、算法、迭代器、函数对象、适配器、内存分配器

2. 容器

1. 容器分类

1. 序列式容器
    - 每个元素都有固定位置 -- 取决于插入的时机和地点，和元素值无关
    - vector，deque，list（列表），stack（栈），queue(队列)
2. 关联式容器
    - 元素的位置取决于特定的排序准测，和插入顺序无关
    - set,multiset,map,multimap

数据结构	描述	实现头文件
向量（vector）	连续存储的元素	
列表（list）	有节点构成的双向链表，每个结点包含着一个元素	
双队列（deque）	连续存储的指向不同元素的指针所组成的数组	
集合（set）	由节点组成的红黑树，每个节点都包含着一个元素，节点之间以某种作用于元素对的谓词排列，没有两个不同的元素能够拥有相同的次序即不可能出现两个相同的元素	
多重集合（multiset）	允许存在两个次序相等的元素的集合	
栈（stack）	后进先出的值的排列	
队列（queue）	先进先出的值的排列	
优先队列（priority_queue）	元素的次序是由作用于所存储的值对上的某种谓词决定的一种队列	
映射(map)	由{键，值} 对组成的集合，以某种作用于键对上的谓词排列	
多重映射（multimap）	允许键对有相等的次序的映射	

1. vector 容器

1. vector 容器简介

- vector 是将元素置于一个动态数组中加以管理的容器
- vector 可以随机存取元素（支持索引值直接存取，用[]操作符或at()方法
- vector 尾部添加或移除元素非常快速，但是在中部或头部插入元素或移除元素比较费时

2. vector 对象的默认构造

***vector*采用模板类实现，*vector*对象的默认构造形式**

```
vector<T> vecT

vector<int> vecInt
vector<float> vecFloat
vector<string> vecString

class CA{}
vector<CA*> vecpCA //用于存储CA对象的指针的vector容器
vector<CA> vecCA //用于存放CA对象的vector 容器，由于容器元素的存放是按值赋值的方式进行的
                 // 所以此时CA必须提供CA的拷贝构造函数，以保证CA对象间拷贝正常
```

3. vecrtor对象的带参数构造

理论知识

- vector(beg,end)　构造函数将 [beg,end) 区间中的元素拷贝给本身，注意该区间是左闭右开的区间
- vector(n,elem)　构造函数将n个elem拷贝给本身
- vector(const vector &vec) 拷贝构造函数

```
int iArrary[]={0,1,2,3,4}

vector<int> v1(iArry,iArry+5)
vector<int> v2(3,10)    //存贮3个10

vector<int> v3(v1) // 触发拷贝构造函数，用已经初始化的v1 构造v3  v3 的内容和v1 相同
```

4. vector 的赋值

理论知识

- vector.assign (beg,end)　将 [beg,end) 区间中的元素拷贝给本身，注意该区间是左闭右开的区间指针
- vector.assign (n,elem)　将 n个elem 拷贝赋值给本身
- vector& operator= (const vector &vec)　重载等号操作符，在使用v1 = v2 时触发
- vector.swap(vec) 将vec于本身的元素互换

```
vector<int> vecIntA, vecIntB,vecIntC,vecIntD;

int iArray[] = {0,1,2,3,4};

vecIntA.assign(iArry,iArry+5); // 不管vecIntA 中原来有多少个数据，全部清除，后再将iArray数组中的数据存储进去

vecIntB.assign(vecIntA.begin(),vecIntA,end()); //vecIntA.begin() 指向第0个元素，vecIntA,end() 指向最后一个元素的下一个元素
vecIntC.assign(4,10);// 拷贝4个10到vecIntC中

vecIntB.swap(vecIntC);     // vecIntB和vecIntC 中的元素进行交换


```

5. vector的大小

理论知识

- vector.size() 返回容器中元素的个数
- vector.empty() 判断容器是否为空　空= true
- vector.resize(num) 重新指定容器的长度为num，若容器变长，则以默认值填充新位置，若容器变短，则末尾超出容器长度的元素被删除
- vector.resize(num,elem) 重新指定容器的长度为num, 容容器边长，则以elem 值填充新位置，若容器变短，则超出容器长度的元素被删除

6. vector 末尾的添加移除操作

- vector veclnt;
    - veclnt.push_back(1)　尾部加入一个元素 1
    - veclint.pop_back()　删除末尾的元素

7. vector 数据的获取

理论知识;

- vec.at(idx) 返回索引 idx 所指的数据，抛出out_of_range异常
- vec[idx] 返回索引idx所指向的数据，越界时，运行直接报错

```
vector<int> vecInt;
vecInt.assign(3,10);
vecInt.at(4)=100; //直接抛出异常  抛出 out_of_range 异常
vecInt.at(2)= vecInt[2];
```

8. vector 的插入

理论知识

- vector.insert(pos,elem) 在 pos的位置插入一个elem元素的拷贝，返回新数据的位置 pos为指针， v1.begin() +x
- vector.insert(pos,n,elem) 在pos 位置插入n 个elem 数据，无返回值
- vector.insert (pos,beg,end) 在pos位置插入[beg, end) 区间的数据，无返回值

2. 迭代器

1. 迭代器的基本概念

- 迭代器时一种检查容器内元素并且遍历容器内元素的数据类型
- 迭代器提供对一个容器的对象的访问方法，并且定义了容器中对象的范围

2. vector 容器中迭代器的使用

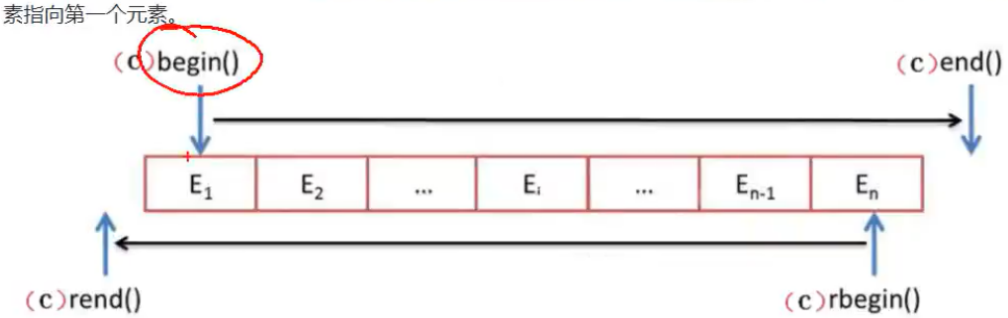
vector::iterator iter // 变量名为iter

vector 容器的迭代器属于 “随机访问迭代器”，迭代器一次可以移动多个位置

成员函数	功能
begin()	返回指向容器中第一个元素的正向迭代器，如果是 const 类型容器，在该函数返回的是常量正向迭代器
end()	返回指向元素最后一个元素之后一个位置的正向迭代器，
rbegin()	返回指向最后一个元素的反向迭代器，如果是const 类型容器，在该函数返回的是常量反向迭代器
rend()	返回指向第一个元素之前一个位置的反向迭代器，如果是const类型的容器，在该函数返回的是常量反向迭代器，此函数通常和rbegin() 搭配使用
cbegin()	和begin() 功能类似，只不过其返回的迭代器类型为常量正向迭代器，不能用于修改元素
cend()	和end() 功能相同，只不过其返回的迭代器类型为常量正向迭代器，不能用于修改元素
crbegin()	和rbegin() 功能相同，只不过其返回的迭代器类型为常量反向迭代器，不能用于修改元素
crend()	和rend() 功能相同，只不过其返回的迭代器类型为常量反向迭代器，不能用于修改元素

4）begin和end操作

每种容器都定义了一队命名为begin和end的函数，用于返回迭代器。如果容器中有元素的话，由begin返回的元素指向第一个元素。




```
#include <iostream>
#include <vector>

using namespace std;

int main() {
    vector<int> vecIntA;
    int iArray[]={1,2,3,4};
    // 构造一个迭代器对象
    vector<int> ::iterator it;

```

```
    it=vecIntA.begin();

    cout<< *it<<endl;   // it 不是一个指针，而是进行了一个* 重载

    for(it = vecIntA.begin();it!= vecIntA.end();it++)
        cout << *it;
    cout<<endl;


    return 0;


}


```

3. 迭代器失效

- 插入元素后失效
- 删除元素后失效

3. deque 容器

1. deque 简介

- deque 是 “ double-ended queue" 的缩写，和vector 一样都是STL的容器
- deque 是双端数组而vector是单端的
- deque 在接口上和vector非常相似，在许多操作的地方可以直接替换
- deque可以随机存取元素（支持索引值直接存取，用[]操作符或者at() 方法）
- deque头部和尾部或者移除元素都非常快速，但是在中部安插元素或移除元素比较费时
- #include

2. deque 容器的操作

- deque 和vector 在操作上几乎一样，deque多两个函数
- deque.push_front(elem) 在容器头部插入一个数据
- deque.push_frount() 删除容器的第一个数据
- deque.erase(deque.begin()) 删除容器第一个数据

- vector , deque 相当于动态数组

4. list 容器

查找 删除 时间复杂度都是 O(n)

vector ,deque 查找 删除时 O(1)

优势在 插入和删除元素效率高

1. list 容器简介

- list 时一个双向链表容器，可高效的进行插入删除元素
- list 不可以随机存取元素，所以不支持at.(pos) 函数与[] 操作符
  - it++ 可以
  - it+ 5 不可以
- #include

2. list对象的默认构造

- list 采用模板类实现，对象的默认构造形式： list lst

```
list<int> lstInt;
list<float> lstFloat;
list<string> lstString;
```

3. list头尾的添加移除操作

- list.push_back(elem) 在尾部加入一个元素
- list.push_front(elem) 在头部插入一个元素
- list.pop_back() 删除容器最后一个元素
- list.pop_front() 删除容器第一个元素
- list.front() 返回容器第一个元素
- list.back() 返回容器最后一个元素
- 可遍历
- stack 和 queue 不可遍历

4.list 的迭代器

- list 容器的迭代器是 ” 双向迭代器“ ， 双向迭代器从两个方向读写容器，除了提供前向迭代器的全部操作之外，双向迭代器还提供前置和后置的自减运算
- list.begin() 返回容器中第一个元素的迭代器
- list.end() 返回容器中最后一个元素之后的迭代器
-
- list.rbegin() 返回容器中倒数第一个元素的迭代器 反向迭代器，++ 先前走
- list.rend() 返回容器中倒数最后一个元素后面的迭代器 ， 可理解为 第一个元素的前一个的迭代器

5. list 对象的带参数构造

- list(n,elem) 构造函数讲n个elem 拷贝给本身
- list(beg,end) 构造函数讲[beg,end） 区间中的元素拷贝给自身
- list(const list &lst) 拷贝构造函数

6. list的赋值

- 对象已经构造好了
- list.assign(beg,end) 将区间[beg,end) 区间中的数据拷贝赋值给本身，注意该区间是左闭右开的区间
- list.assign(n,elem) 将n个elem 拷贝赋值给本身
- list & operator= (const list &lst) 重载等号操作符
- list.swap(lst) //将lst 与本身list 中的元素互换

```
list<int> lstIntA,lstIntB,lstIntC;
lstIntA.push_back(1);
lsInt.push_back(2);

lsIntB.assgin(lstIntA.begin(),lstIntA.end());
lstIntC.assgin(5,8); // 8 8 8 8 8
lstIntB=lsIntC;
lsIntC.swap(lstIntA);
```

7. list 容器的大小

- list.size() 返回容器中元素的个数
- list.empty() 判断容器是否为空
- list.resize(num) 重新指定容器的长度为num ,若容器变长，则以默认值填充新位置，若容器变短，则末尾超出容器长度的元素被删除
- list.resize(num,elem) 重新指定容器长度为num,若容器变长，则以elem 值填充新位置，如果容器变短，则末尾超出容器长度的元素被删除

8. list 插入

- list.insert(pos,elem); 在pos 位置插入一个elem 元素的拷贝，返回新数据的位置
- list,insert(pos,n,elem) 在pos位置插入n个elem 数据，无返回值
- list,insert(pos,beg,end) 在pos位置插入[beg,end) 区间的数据，无返回值
- pos 是迭代器，不能随意插入数字

9. list 删除

- list.clear() 移除容器中所有的数据
- list.erase(beg,end) 删除[bed,end) 区间的数据，返回下一个数据的位置
- list.erase(pos) 删除pos 位置的数据，返回下一个数据的位置 返回新的迭代器
- list.remove(elem) 删除容器中所有与elem 值匹配的元素

10. list 反转

- list.reverse() 反转 list 中的数据 1 2 3 4 变成 4 3 2 1

11. list 迭代器失效

- 删除结点导致迭代器失效

5. stack 容器

1. stack 简介

- stack 是堆栈容器，是一种先进后出的 容器
- #include

2. stack 对象的默认构造

- stack 采用模板类实现，stack 对象的默认构造形式为stack s ;
- stack stkInt; 一个存放in 的stack 容器

3. stack 的push() 和pop() 方法

- stack.push(elem) 往栈头添加元素
- stack.pop() 从栈头移除第一个元素
- stack 容器没有迭代器，因为栈不允许遍历
- stack.top() 只返回栈顶元素
- stack.empty() 空返回true 不空 false

4 . stack 对象的拷贝构造与赋值

- stack(const stack &stk) 拷贝构造函数
- stack& ioerator = (const stack &stk); 重载等号操作符

```
stack <int> stk2(stk);
stack<int> stk3=stk; // 调用拷贝构造函数
stk3=stk;  //调用= 重载
```

5. stack 的大小

- stack.empty() 判断堆栈是否为空
- stack.size() 返回堆栈的大小

6. queue 容器

1. queue 简介

- queue 是队列容器，是一种先进先出的容器
- #include
- 没有提供迭代器，不能够遍历

```
q1.push(1);  // 入队
q1.pop(); //删除队首元素
```

2. queue 容器对象的拷贝构造和赋值

- queue(const queue &que) 拷贝构造函数
- queue& operator=(const queue &que) 重载等号操作符

```
queue<int> queIntA;
queIntA.push(1);
queIntA.push(2);


queue<int> queIntB(queIntA); //拷贝构造
queue<int< queIntC;
queIntC= queIntA; //赋值

```

- stack ,queue 序列容器 元素的位置和插入元素的位置和时机有关系，和元素大小无关

7. Set 和multiset 容器

- set 和map 是关联容器 元素的位置 和元素的大小有关，和时机无关

1. set/multiset 容器简介

- set是一个集合容器，其中包含的元素是唯一的，集合中的元素按照一定的顺序排列，元素插入的过程是按排序规则插入（log2(n)），所以不能指定插入位置
- set采用红黑树变体的数据结构实现，红黑树属于平衡二叉树，在插入操作和删除操作上比vector快，
- set 不可以直接存取元素 （不可以使用at.(pos) 与[] 操作符）
- multiset 与set的区别: set 支持唯一键值，每个元素值只能出现一次，而multiset 中同一值可以出现多次
- 不可以直接修改set或multiset 容器中的元素值，因为该容器是自动排序的，如果希望修改一个元素值，必须先删除原有的元素，再插入新的元素
- #include
- set 默认升序排序

2. set/multiset 容器对象的默认构造

```
set<int> setInt;

multiset<int> mulsetInt;
```

3. set 容器的插入与迭代器

- set.insert(elem) 在容器中插入元素
- set.begin() 返回容器中第一个数据的迭代器
- set.end() 返回容器中最后一个数据之后的迭代器
- set.rbegin() 返回容器倒数第一个元素的迭代器
- set.rend() 返回容器中倒数最后一个元素后面的迭代器

```
set<int>::iterator it;
for（it=s1.begin() ;it!=s1.end();it++）
        cout<<*it<<cout <<endL;
set<int>::reverse_iterator it2;
for （it2= s1.rbegin() ;it2 != s1.rend();it2++)
       cout<<*it2<<cout << endL;



  return e
```

```
1 2 3 4
4 3 2 1
```

4. set 容器拷贝与赋值

- set(const set& st) 拷贝构造函数
- set& operator = (const set &st)
- set.swap(st) // 交换两个集合容器
- set.seize()
- set.clear()
- set.erase(pos)
- set.erase(beg,end)
- set.erase(elem) 删除容器中值为elem 的数据

5. set 容器的元素排序

- set<int,less> setIntA 该容器是升序方式排列
- set<int,greater> setIntB 该容器时按照降序方式排列
- set 相当于set<int, less>
-

6. set 容器的查找

- set.fine(elem) 查找elem 元素，返回指向elem 元素的迭代器
- set.couont(elem) 返回容器中值为elem 的元素个数，对于set 来说 要么是 0 要么是 1 对于multiset 来说值可能大于1
- set.lower_bound(elem) 返回第一个 >= elem元素的迭代器
- set. upper_bound(elem) 返回第一个 >elem 元素的迭代器

7. set.equal_range(elem)

- 返回容器中与elem 相等的上下限的两个迭代器，上限是闭区间，下限是开区间 如[beg,end)
- 函数返回两个迭代器 ，而这两个迭代器被封装在pair中

```
pair<set<int> ::iterator,set<int>::iterator> pairIt=
setInt.equal_range(5);
```

- pair<T1,T2> 存放的两个值的类型，可以不一样 如T1 为int ,T2 为float T1 T2 也可以是自定义类型
- pair.fist是pair 里面的第一个值，是T1 类型
- pair.second 是pair 里面的第二个值，是T2 类型

8 map 容器

1. map/multimap 容器对象的默认构造

- map /multimap 采用模板类实现，对象的默认构造形式

```
 map<T1,T2> mapTT;
multimap<T1,T2> multimap TT;
// 如
map<int,char> mapA;
map<string,float> mapB;
```

2. map 容器的插入

- map.insert(...) 往容器中插入元素，返回pair
- mapStu.insert(pair<int,string>(3,"小张"))
- 通过value_type 的方式插入对象:mapStu.insert(pair<int,string>::value_type(3,"小张"))

3. map容器对象获取建对应的值

- 方法1： 使用[]
- 方法2 使用find() 成功返回对应的迭代器，失败 返回end() 的返回值

```
map<int,string>::iterator it = mapS.find(3);
```

- 如果使用at() 函数，如果键值对不存在则会抛出 out_of_range 异常