# STL适配器总结

## 1. 容器适配器





**通用操作（初始化+关系运算）**

* 默认基础容器类型初始化  
  基本初始化方式：
  + 空适配器：A a;
  + 复制容器c元素的适配器：A a(c);

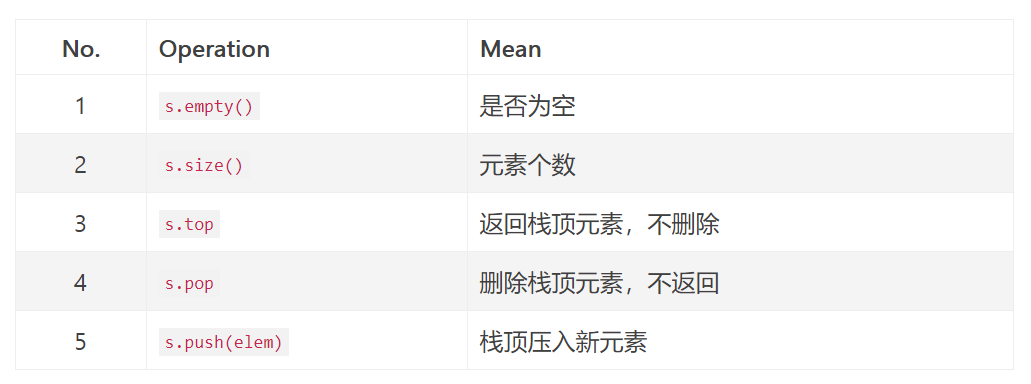
|  |
| --- |
| stack<int> oTestStk1;// 等同于stack<int,deque<int> > oTestStk1;  stack<int> oTestStk2(dep);// 等同于stack<int,deque<int> > oTestStk2(dep); |

覆盖基础容器类型初始化  
覆盖上面基础容器类型，例如：利用vector<int>覆盖stack默认的deque<int>

|  |
| --- |
| stack<int,vector<int> > oTestStk1;  stack<int,vector<int> > oTestStk2(vec); |

关系运算  
支持关系运算：==,!=,>,>=,<,<=  
使用关系运算要求元素支持==和<两个运算符。

* **栈适配器**
* stack的基本操作  
  stack是后进先出LIFO



虽然stack的默认基础容器是deque,但是，不再支持deque独有的方法。例如，stack不再支持deque的push\_back方法，取而代之的是push方法。

**队列适配器**

queue的基本操作  
queue是后进先出FIFO



虽然queue的默认基础容器是deque,但是，不再支持deque独有的方法。例如，queue不再支持deque的push\_back方法，取而代之的是push方法。

**2. 函数对象适配器**

函数对象的适配器，用于特化和扩展一元或者二元函数对象，主要分为两类：

1. 绑定器(binder)：给二元函数对象绑定一个常量，转化成一元函数对象。包括bind1st和bind2nd。
2. 取反器(negator)：将谓词函数对象结果取反。包括not1和not2。



**绑定STL的仿函数**

|  |
| --- |
| **#include <iostream>**  **#include <functional>**  **#include <algorithm>**  **using namespace std;**  **int main () {**  **int numbers[] = {1,2,3,4,5,6};**  **int cnt = count\_if (numbers, numbers+6, bind1st(less<int>(),4) );**  **cout << "There are elements that are > 4:" << cnt << "\n";**  **cnt = count\_if (numbers, numbers+6, bind2nd(less<int>(),4) );**  **cout << "There are elements that are < 4:" << cnt << "\n";**  **nt = count\_if (numbers, numbers+6, bind2nd(equal\_to<int>(),4) );**  **cout << "There are elements that are = 4:" << cnt << "\n";**  **return 0;**  **}** |

**绑定自定义的仿函数**

|  |
| --- |
| **#include <iostream>**  **#include <functional>**  **#include <algorithm>**  **using namespace std;**  **class Functor:public binary\_function<int,int,void>{**  **public:**  **void operator()(int first, int secend)const{**  **cout << "first:" << first << " " << "secend:" << secend << endl;**  **return;**  **}**  **};**  **int main () {**  **// binder1st<Functor> fctor = bind1st(Functor(),1);**  **// fctor(2);**  **bind1st(Functor(),1)(2);**  **bind2nd(Functor(),1)(2);**  **return 0;**  **}** |

## 3. 迭代器适配器

### 分类

* 流迭代器适配器
* 输出流迭代器适配器ostreambuf\_iterator
* 输入流迭代器适配器istreambuf\_iterator
* 逆向迭代器适配器reverse\_iterator
* 插入迭代器适配器
* 插入迭代器适配器insert\_iterator
* 前向插入迭代器适配器front\_insert\_iterator
* 后向插入迭代器适配器back\_insert\_iterator

## 1. 流迭代器适配器

### 1.2 输出流迭代器适配器ostreambuf\_iterator

* ostreambuf\_iterator.cpp

|  |
| --- |
| **#include <iostream>**  **#include <iterator>**  **#include <vector>**  **using namespace std;**  **int main(){**  **int arr[] = {1,2,3,4,5,6};**  **ostream\_iterator<int> os\_it(cout," ");**  **copy(arr,arr+6,os\_it);**  **vector<int> vec(arr,arr+6);**  **ostream\_iterator<int> os\_it2(cout," ");**  **copy(vec.begin(),vec.end(),os\_it2);**  **}** |

### 1.2 输入流迭代器适配器istream\_iterator

* istream\_iterator.cpp

|  |
| --- |
| **#include <iostream>**  **#include <vector>**  **#include <iterator>**  **using namespace std;**  **int main(){**  **istream\_iterator<int> eos;**  **istream\_iterator<int> is\_it(cin);**  **ostream\_iterator<int> os\_it(cout,",");**    **int arr[10];**  **copy(is\_it,eos,arr);**  **copy(arr,arr+10,os\_it);**  **cout << endl;**    **cin.clear();**  **vector<int> vec;**  **copy(istream\_iterator<int>(cin),eos,back\_inserter(vec));**  **copy(vec.begin(),vec.end(),os\_it);**  **}** |

**2. 逆向迭代器适配器reverse\_iterator**

逆向迭代器适配器:template <class Iterator> class reverse\_iterator;

* reverse\_iterator.cpp

|  |
| --- |
| #include <iostream>  #include <vector>  #include <iterator>  using namespace std;  int main(){  int arr[] = {1,2,3,4,5,6};  vector<int> vec(arr,arr+6);  copy(vec.begin(),vec.end(),ostream\_iterator<int>(cout,","));  cout << endl;  copy(  reverse\_iterator<vector<int>::iterator>(vec.end()),  reverse\_iterator<vector<int>::iterator>(vec.begin()),  ostream\_iterator<int>(cout,","));  cout << endl;  } |

**3. 插入迭代器适配器**

|函数|作用|  
|:-|  
|insert\_iterator<Container> inserter (Container& x, Iterator it)|创建插入迭代器,必须是包含成员函数insert()的容器，比如：顺序容器(vector、list与deque)、关联容器(map与set)|  
|front\_insert\_iterator<Container> front\_inserter (Container& x)|创建前向插入迭代器，必须是包含成员函数push\_front()的容器，比如：list与deque|  
|back\_insert\_iterator<Container> back\_inserter (Container& x)|创建后向插入迭代器，必须是包含成员函数push\_back()的容器，比如：vector、list与deque|

inserter.cpp

|  |
| --- |
| #include <iostream>  #include <vector>  #include <iterator>  using namespace std;  int main(){  int arr[] = {1,2,3,4,5,6};  vector<int> vec(arr,arr+6);  vector<int>:: iterator it = vec.begin();  advance(it,3);  copy(arr,arr+3,it);  copy(vec.begin(),vec.end(),ostream\_iterator<int>(cout,","));  cout << endl;  copy(arr+3,arr+6,inserter(vec,it));  copy(vec.begin(),vec.end(),ostream\_iterator<int>(cout,","));  } |

front\_inserter.cpp

|  |
| --- |
| #include <iostream>  #include <list>  #include <iterator>  using namespace std;  int main(){  int arr[] = {1,2,3,4,5,7};  list<int> vec(5);  ostream\_iterator<int> os\_it(cout,",");  copy(arr,arr+3,front\_inserter(vec));  copy(vec.begin(),vec.end(),os\_it);  } |

back\_inserter.cpp

|  |
| --- |
| #include <iostream>  #include <list>  #include <iterator>  using namespace std;  int main(){  int arr[] = {1,2,3,4,5,7};  list<int> vec(5);  ostream\_iterator<int> os\_it(cout,",");  copy(arr+3,arr+6,back\_inserter(vec));  copy(vec.begin(),vec.end(),os\_it);  } |