For\_each

遍历算法 遍历容器元素

@param beg 开始迭代器

@param end 结束迭代器

@param \_callback 函数回调或者函数对象

@return 函数对象

使用函数，仿函数都可以作为遍历的规则

Void myPrint(int v)

{  
cout <<v << endl;

}

Class myPrint01

{

Public:

Void operator()(int v)

{  
cout <<v <<endl;

}

}

Void test()

{  
vector<int> v;

For(int I = 0 ; I < 10; ++i)

{  
v.push\_back(i);

}

For\_each(v.begin(), v.end(), myPrint);

For\_each(v.begin(), v.end(), myPrint01());

}

有返回值， 类中存储信息

Class myPrint

{

Public:

Void operator()(int v)

{  
cout <<v <<endl;

}

Int m\_Count;

}

Void test()

{  
vector<int> v;

For(int I = 0; I < 10; ++i)

{

v.push\_back(i);

}

//for\_each返回函数对象，其中可以存储信息

myPrint print = for\_each(v.begin(), v.end(), myPrint());

cout << print.m\_Count << endl;  
}

可以绑定参数进行输出， 适配器

Class myPrint : public binary\_function<int, int , void>  
{  
public:

Void operator()(int val) const

{  
cout << val << endl

}

}

Void test()

{  
vector<int> v;

For(int I = 0; I <10; ++i)

{

v.push\_back(i);

}

For\_each(v.begin(), v.end(), bind2nd(myPrint(),100) );

}

transform算法

将指定区间元素搬运到另一容器中

不会分配内存，需要我们手动分配

@param beg1 源容器开始迭代器

@param end1 源容器结束迭代器

@param beg2 目标容器开始迭代器

@param \_cakkback 回调函数或者函数对象

@return 返回目标容器迭代器

Class Transform

{  
public:

Int operator() (int val)

{  
return val + 10;  
}

}

Void test()

{

Vector<int> v; //原容器

For(int I = 0; I < 10; ++i)

{  
v.push\_back(i);

}

Vector<int> vTarget; //目标容器

vTarget.resize(v.size()); //开辟空间

Transform(v.begin(), v.end(), vTarget.begin(), Transform());

}

Transform可以将两个容器的数据运算后搬运到目标容器中

Class Transform

{  
public:

Int operator() (int val, int val2)

{  
return val + val2;  
}

}

Void test()

{

Vector<int> v1;

Vector<int> v2;

For(int I = 0; I < 10; ++i)

{  
 v1.push\_back(100 + i);

V2.push\_back(200 + i);

}

Vector<int> vTarget;

vTarget.resize(v1.size());

transform(v1.begin(), v1.end(), v2.begin(), v2.end(), vTarget.begin(), Transform());

}