函数对象

http://www.cppblog.com/mzty/archive/2005/12/14/1746.html

函数指针的一种替代策略是Function object（函数对象）。

函数对象与函数指针相比较有两个方面的优点：首先如果被重载的调用操作符是inline函数则编译器能够执行内联编译，提供可能的性能好处；其次函数对象可以拥有任意数目的额外数据，用这些数据可以缓冲结果，也可以缓冲有助于当前操作的数据。

函数对象是一个类，它重载了函数调用操作符operator() ,该操作符封装了一个函数的功能。典型情况下函数对象被作为实参传递给泛型算法，当然我们也可以定义独立的函数对象实例。

来看下面的二个例子： 比较理解会更好些：

```
#include<vector>
#include<string>
#include<iostream>
#include<algorithm>
using namespace std;
class Sum {
int val;
public:
Sum(int i) :val(i) { }

//当在需要int的地方，Sum将自动转换为int类型
//这里是为了方便cout<<Sum的实例；
operator int() const { return val; }

//写在类中的函数代码一般默认为内联代码
int operator()(int i) { return val+=i; }
};

void f(vector<int> v)
{
Sum s = 0; //Sum s = 0等价于Sum s(0),不等价于Sum s;s = 0;

//对vector<int>中的元素求和
//函数对象被作为实参传递给泛型算法
s = for_each(v.begin(), v.end(), s);

cout << "the sum is " << s << "\n";

//更简单的写法，定义独立的函数对象实例
cout << "the sum is " << for_each(v.begin(), v.end(), Sum(0)) << "\n";
}


int main()
{
vector<int> v;
v.push_back(3); v.push_back(2); v.push_back(1);
f(v);
system("pause");
return 0;
}
```

——

```
#include <iostream>
#include <list>
#include <algorithm>
#include "print.hpp"
using namespace std;

// function object that adds the value with which it is initialized
class AddValue {
  private:
    int theValue;    // the value to add
  public:
    // constructor initializes the value to add
    AddValue(int v) : theValue(v) {
    }
```

```
    // the ``function call'' for the element adds the value
    void operator() (int& elem) const {
        elem += theValue;
    }
};

int main()
{
    list<int> coll;

    // insert elements from 1 to 9
    for (int i=1; i<=9; ++i) {
        coll.push_back(i);
    }

    PRINT_ELEMENTS(coll,"initialized:               ");

    // add value 10 to each element
    for_each (coll.begin(), coll.end(),    // range
              AddValue(10));               // operation

    PRINT_ELEMENTS(coll,"after adding 10:           ");

    // add value of first element to each element
    for_each (coll.begin(), coll.end(),    // range
              AddValue(*coll.begin()));    // operation

    PRINT_ELEMENTS(coll,"after adding first element: ");
}
```

---

operator()中的参数为container中的元素

————————————————————————————

另外的实例：

Function Objects as Sorting Criteria

Programmers often need a sorted collection of elements that have a special class (for example, a collection of `persons`). However, you either don't want to use or you can't use the usual operator < to sort the objects. Instead, you sort the objects according to a special sorting criterion based on some member function. In this regard, a function object can help. Consider the following example:

```
// fo/sortl.cpp

    #include <iostream>
    #include <string>
    #include <set>
    #include <algorithm>
    using namespace std;


    class Person {
      public:
        string firstname() const;
        string lastname() const;
        ...
    };


    /* class for function predicate* - operator() returns whether a person is less than another person*/
    class PersonSortCriterion {
      public:
        bool operator() (const Person& p1, const Person& p2) const {
            /* a person is less than another person* - if the last name is less* - if the last name is equ
al and the first name is less*/
            return p1.lastname()<p2.lastname() ||
                   (! (p2.lastname()<p1.lastname()) &&
                    p1.firstname()<p2.firstname());
        }
    };


    int main()
    {

        //declare set type with special sorting criterion
```

```
    typedef set<Person,PersonSortCriterion> PersonSet;

    //create such a collection
    PersonSet coll;
    ...


    //do something with the elements
    PersonSet::iterator pos;
    for (pos = coll.begin(); pos != coll.end();++pos) {
        ...
    }
    ...
}



//fo/foreach3.cpp

#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;


//function object to process the mean value
class MeanValue {
  private:
    long num;     //number of elements
    long sum;     //sum of all element values
  public:
    //constructor
    MeanValue() : num(0), sum(0) {
    }


    //"function call"//-process one more element of the sequence
    void operator() (int elem) {
        num++;          //increment count
        sum += elem;    //add value
    }


    //return mean value
    double value() {
        return static_cast<double>(sum) / static_cast<double>(num);
    }
};

int  main()
{
     vector<int> coll;


     //insert elments from 1 to 8
     for (int i=1; i<=8; ++i) {
         coll.push_back(i);
     }


     //process and print mean value
     MeanValue mv = for_each (coll.begin(), coll.end(), //range
                              MeanValue());             //operation
     cout << "mean value: " << mv.value() << endl;
}
```