C++语言程序设计

贺利坚 主讲

函数适配器

为什么要用函数适配器

- 例：有

```
int intArr[8] = {30, 90, 10, 40, 70, 50, 20, 80 };
vector<int> a(intArr, intArr + 8);
```

- 用find_if查找满足第一个大于40的元素

```
vector<int>::iterator p = find_if(a.begin(), a.end(), greater40);
```

- 期望
  - 用一个一般性的函数完成类似greater40的功能
- 通用需求
  - 将一种函数对象，转换为另一种符合要求的函数对象
- 具体要求
  - 对函数返回值进行进一步的简单计算
  - 填上多余参数，再代入算法
- 适配器功能
  - 将一种函数对象，转换为另一种符合要求的函数对象

```
template<class InputIterator, class UnaryPredicate>
InputIterator find_if ( InputIterator first,
                        InputIterator last,
                        UnaryPredicate pred )
```

```
greater<int> g;
bool greater40(int x)
{
    return g(x, 40);
}
```

使用绑定适配器

- 绑定适配器
  - 将n元函数对象的指定参数绑定为一个常数，得到n-1元函数对象

```
#include <functional>
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main()
{
   int intArr[] = { 30, 90, 10, 40, 70, 50, 20, 80 };
   const int N = sizeof(intArr) / sizeof(int);
   vector<int> a(intArr, intArr + N);
   vector<int>::iterator p = find_if(a.begin(), a.end(), bind2nd(greater<int>(), 40));
   if (p == a.end())
      cout << "no element greater than 40" << endl;
   else
      cout << "first element greater than 40 is: " << *p << endl;
   return 0;
}
```

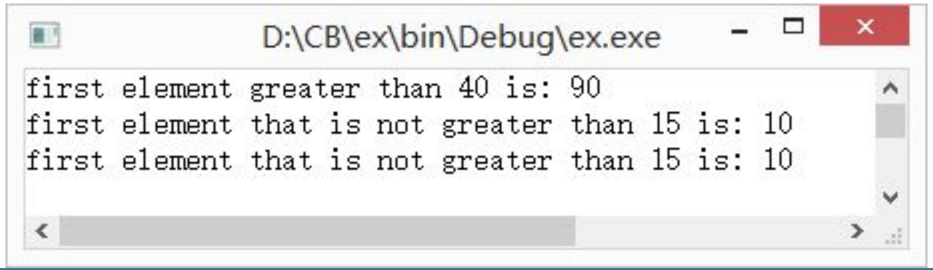
- 绑定适配器
  - 将一个操作数绑定到给定值而将二元函数对象转换为一元函数对象。
- 两个绑定适配器函数
  - bind2nd：将给定值绑定到二元函数对象的第二个实参；
  - bind1st：将给定值绑定到二元函数对象的第一个实参；

STL中函数适配器分类

适配器类型	功能	适配器辅助函数
绑定适配器	将n元函数对象的指定参数绑定为一个常数，得到n-1元函数对象	bind1st bind2nd
组合适配器	将指定谓词的结果取反	not1 not2
指针函数适配器	对一般函数使用，使之能够作为其它函数适配器的输入	ptr_fun
成员函数适配器	对成员函数使用，使之能够作为其它函数适配器的输入	mem_fun mem_fun_ref

函数适配器用法示例

```
#include <functional>
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
bool g(int x, int y)
{
    return x > y;
}
int main()
{
    int intArr[8] = { 30, 90, 10, 40, 70, 50, 20, 80 };
    vector<int> a(intArr, intArr + 8);

    return 0;
}
```


D:\CB\ex\bin\Debug\ex.exe

```
first element greater than 40 is: 90
first element that is not greater than 15 is: 10
first element that is not greater than 15 is: 10
```

ptr_fun(g): 使g能成为bind2nd适配器的输入

```
vector<int>::iterator p;
p = find_if(a.begin(), a.end(), bind2nd(ptr_fun(g), 40));
if (p == a.end())
    cout << "no element greater than 40" << endl;
else
    cout << "first element greater than 40 is: " << *p << endl;
```

not1(...): 生成一元函数的逻辑反

```
p = find_if(a.begin(), a.end(), not1(bind2nd(greater<int>(), 15)));
if (p == a.end())
    cout << "no element is not greater than 15" << endl;
else
cout << "first element that is not greater than 15 is: " << *p << endl;
```

not2(...): 生成二元函数的逻辑反

```
p = find_if(a.begin(), a.end(), bind2nd(not2(greater<int>()), 15));
if (p == a.end())
    cout << "no element is not greater than 15" << endl;
else
    cout << "first element that is not greater than 15 is: " << *p << endl;
```

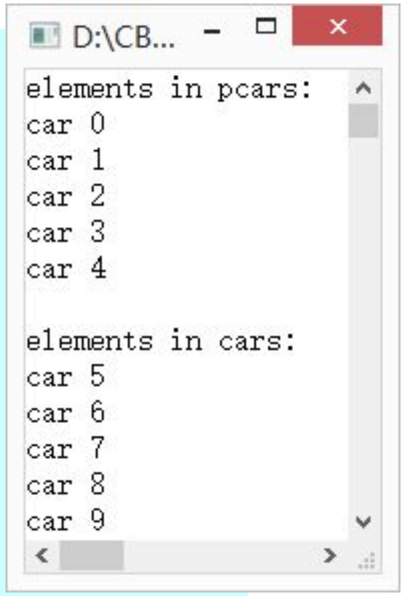
成员函数适配器用法

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

struct Car
{
    int id;
    Car(int id)
    {
        this->id = id;
    }
    void display() const
    {
        cout << "car " << id << endl;
    }
};
```

```
int main()
{
    vector<Car *> pcars;
    vector<Car> cars;
    for (int i = 0; i < 5; i++)
        pcars.push_back(new Car(i));
    for (int i = 5; i < 10; i++)
        cars.push_back(Car(i));

    cout << "elements in pcars: " << endl;
    for_each(pcars.begin(), pcars.end(), mem_fun(&Car::display));
    cout << endl;

    cout << "elements in cars: " << endl;
    for_each(cars.begin(), cars.end(), mem_fun_ref(&Car::display));
    cout << endl;

    for (size_t i = 0; i < pcars.size(); ++i)
        delete pcars[i];

    return 0;
}
```

使成员函数作为函数对象，传入对象指针

使成员函数作为函数对象，传入对象引用



```
elements in pcars:
car 0
car 1
car 2
car 3
car 4

elements in cars:
car 5
car 6
car 7
car 8
car 9
```