C++语言程序设计

贺利坚 主讲

函数对象

函数对象

- 函数对象
  - 一个行为类似函数的对象
  - 可以没有参数，也可以带有若干参数
  - 其功能是获取一个值，或者改变操作的状态。
- 函数对象的形式
  - 普通函数就是函数对象
  - 重载了“()”运算符的类的实例是函数对象

```
#include <iostream>
#include <vector>
#include <iterator>
#include <algorithm>
#include <functional>
using namespace std;
int main()
{
    const int N = 5;
    vector<int> s(N);
    for (int i = 0; i < N; i++)
        cin >> s[i];
    transform(s.begin(), s.end(),  ostream_iterator<int>(cout, " "), negate<int>());
    cout << endl;
    return 0;
}
```

```
template< typename T >
class square
{
public:
    T operator()(T& v)
    {
        return(v * v);
    }
};
```

```
double square(double x)
{
    return x * x;
}
```

改为：square<int>()

改为：square

函数对象用作为另外函数的参数，从而实现“通用函数”

例：定义表示乘法的函数对象——通过定义普通函数

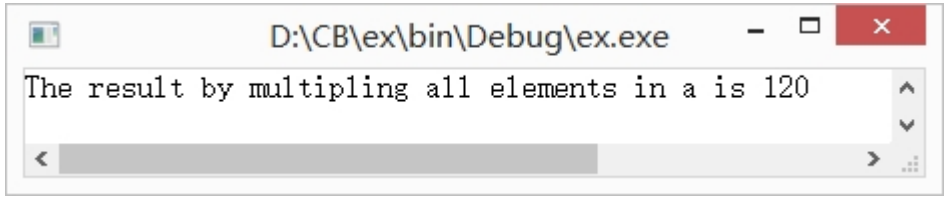
```
#include <iostream>
#include <numeric>
using namespace std;

int mult(int x, int y)
{
    return x * y;
};

int main()
{
    int a[] = { 1, 2, 3, 4, 5 };
    const int N = sizeof(a) / sizeof(int);
    cout << "The result by multipling all elements in a is "
        << accumulate(a, a + N, 1, mult)
        << endl;
    return 0;
}
```

- 累加算法accumulate

  template<class InputIterator, class Type, class BinaryFunction>

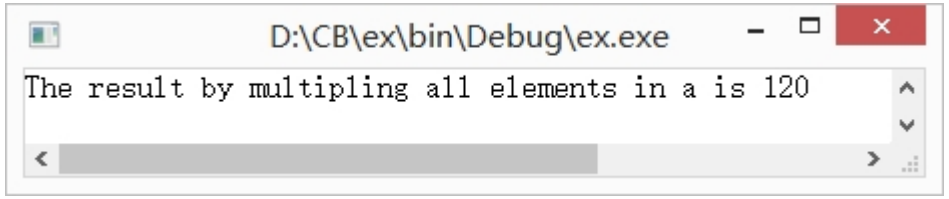
  Type accumulate(InputIterator first, InputIterator last, Type val, BinaryFunction binaryOp);

- 算法功能
  - 对[first, last)区间内的数据进行累“加”
  - val为累“加”的初值
  - binaryOp为用二元函数对象，表示的“加”运算符



定义表示乘法的函数对象——通过重载类的“()”运算符

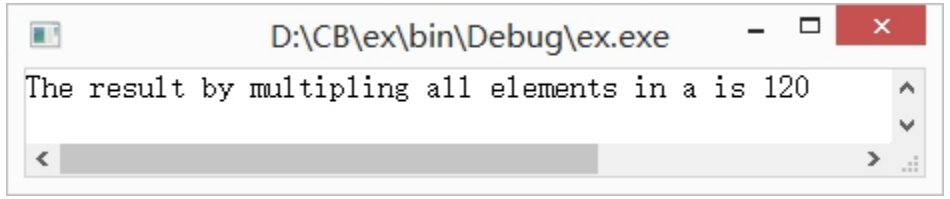
```
#include <iostream>
#include <numeric>
using namespace std;
class MultClass
{
public:
    int operator() (int x, int y) const
    {
        return x * y;
    }
};
int main()
{
    int a[] = { 1, 2, 3, 4, 5 };
    const int N = sizeof(a) / sizeof(int);
    cout << "The result by multipling all elements in a is "
        << accumulate(a, a + N, 1, MultClass()) << endl;
    return 0;
}
```



STL提供的函数对象——#include <functional>

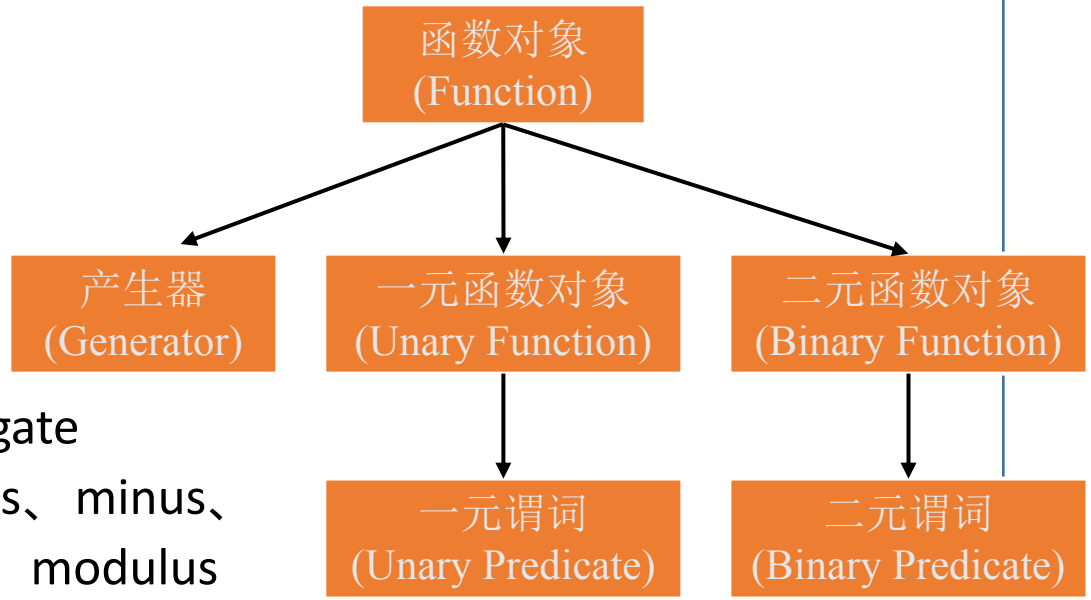
- 用于算术运算的函数对象
  - 一元函数对象：negate
  - 二元函数对象：plus、minus、multiplies、divides、modulus
- 用于逻辑运算的函数对象
  - 一元谓词：logical_not
  - 二元谓词：logical_and、logical_or
- 用于关系运算的函数对象
  - 二元谓词：equal_to、not_equal_to、greater、less、greater_equal、less_equal

```
#include <iostream>
#include <numeric>
#include <functional>
using namespace std;
int main()
{
    int a[] = { 1, 2, 3, 4, 5 };
    const int N = sizeof(a) / sizeof(int);
    cout << "The result by multipling all elements in A is "
        << accumulate(a, a + N, 1, multiplies<int>())
        << endl;
    return 0;
}
```



函数对象概念图

- 一元函数对象：有1个参数的函数对象
- 二元函数对象：有2个参数的函数对象
- 产生器：有0个参数的函数对象
- STL中的函数对象
  - 一元函数对象：negate
  - 二元函数对象：plus、minus、multiplies、divides、modulus



- 累加算法accumulate

```
template<class InputIterator, class Type, class BinaryFunction>
Type accumulate(InputIterator first, InputIterator last, Type val, BinaryFunction binaryOp);
```

```
accumulate(a, a + N, 1, MultClass())
```

```
class MultClass{
public:
    int operator() (int x, int y) const {
        return x * y;
    }
};
```

```
accumulate(a, a + N, 1, mult);
```

应用二元函数对象

```
int mult(int x, int y){
    return x * y;
};
```

```
accumulate(a, a + N, 1, multiplies<int>())
```

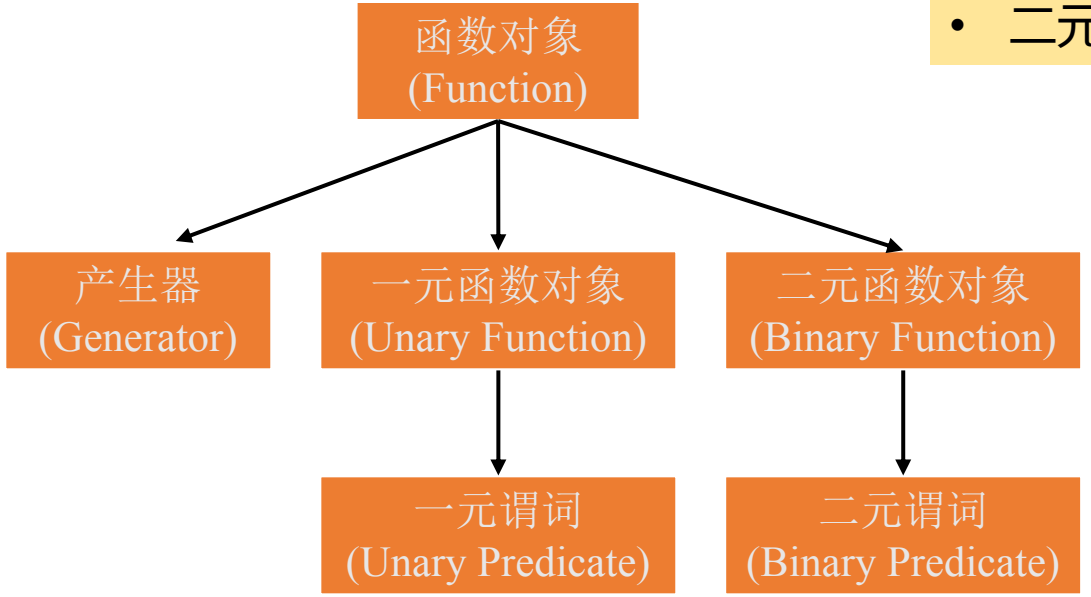
```
transform(s.begin(), s.end(),  ostream_iterator<int>(cout, " "), negate<int>());
```

```
transform(s.begin(), s.end(),  ostream_iterator<int>(cout, " "), square);
```

应用一元函数对象

```
double square(double x){
    return x * x;
}
```

函数对象概念图(谓词)

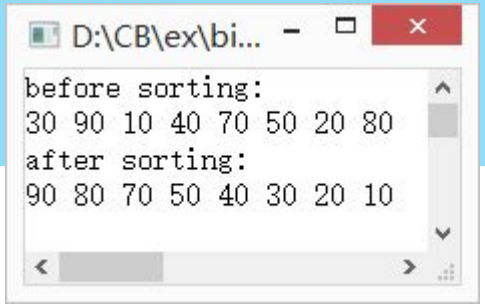


谓词：返回值为bool型的函数对象

- 一元谓词
- 二元谓词

```
#include <iostream>
#include <numeric>
#include <vector>
#include <iterator>
#include <algorithm>
#include <functional>
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);
    cout << "before sorting:" << endl;
    copy(a.begin(), a.end(), ostream_iterator<int>(cout, " "));
    cout << endl;
    sort(a.begin(), a.end(), greater<int>());
    cout << "after sorting:" << endl;
    copy(a.begin(), a.end(), ostream_iterator<int>(cout, " "));
    cout << endl;
    return 0;
}
```



- 用于逻辑运算的函数对象
  - 一元谓词：logical_not
  - 二元谓词：logical_and、logical_or
- 用于关系运算的函数对象
  - 二元谓词：equal_to、not_equal_to、greater、less、greater_equal、less_equal