程序设计实习

郭炜　微博 http://weibo.com/guoweiofpku
http://blog.sina.com.cn/u/3266490431
刘家瑛 微博 http://weibo.com/pkuliujiaying

C++11特性

统一的初始化方法

```
int arr[3]{1, 2, 3};
vector<int> iv{1, 2, 3};
map<int, string> mp{{1, "a"}, {2, "b"}};
string str{"Hello World"};
int * p = new int[20]{1,2,3};

struct A {
        int i,j;  A(int m,int n):i(m),j(n) {     }
};
A func(int m,int n ) { return {m,n}; }
int main() { A * pa = new A {3,7}; }
```

成员变量默认初始值

```
class B
{
        public:
        int m = 1234;
        int n;
};
int main()
{
        B b;
        cout << b.m << endl;  //输出 1234
        return 0;
}
```

auto关键字

用于定义变量，编译器可以自动判断变量的类型

```
auto i = 100;        // i 是 int
auto p = new A(); // p 是 A *
auto k = 34343LL;      // k 是 long long

map<string,int,greater<string> > mp;
for( auto i = mp.begin(); i != mp.end(); ++i)
            cout << i->first << "," << i->second ;
//i的类型是：  map<string,int,greater<string> >::iterator
```

auto关键字

```
class A { };
A operator + ( int n,const A & a)
{
        return a;
}
template <class T1, class T2>
auto add(T1 x, T2 y) -> decltype(x + y)  {
    return x+y;
}

auto d = add(100,1.5); // d是double d=101.5
auto k = add(100,A()); // d是A类型
```

decltype 关键字

求表达式的类型

```
int i;
double t;
struct A { double x; };
const A* a = new A();

decltype(a)      x1;  // x1 is A *
decltype(i)      x2;  // x2 is int
decltype(a->x)   x3;  // x3 is double
decltype((a->x)) x4 = t;  // x4 is double&
```

智能指针shared_ptr

- 头文件： <memory>
- 通过shared_ptr的构造函数，可以让shared_ptr对象托管一个new运算符返回的指针，写法如下：
- shared_ptr<T> ptr(new T); // T 可以是 int ,char, 类名等各种类型

此后ptr就可以像 T* 类型的指针一样来使用，即 *ptr 就是用new动态分配的那个对象，而且不必操心释放内存的事。

- 多个shared_ptr对象可以同时托管一个指针，系统会维护一个托管计数。当无shared_ptr托管该指针时，delete该指针。
- shared_ptr对象不能托管指向动态分配的数组的指针，否则程序运行会出错

智能指针shared_ptr

```
#include <memory>
#include <iostream>
using namespace std;
struct A   {
                int n;
                A(int v = 0):n(v){ }
                ~A() { cout << n << " destructor" << endl; }
};
int main()
{
                shared_ptr<A> sp1(new A(2)); //sp1托管A(2)
                shared_ptr<A> sp2(sp1);          //sp2也托管 A(2)
                cout << "1)" << sp1->n  << "," << sp2->n << endl; //输出1)2,2
                shared_ptr<A> sp3;
                A * p = sp1.get();     //p 指向 A(2)
                cout << "2)" << p->n << endl;
```

输出结果：
1)2,2
2)2

```
    sp3 = sp1;  //sp3也托管 A(2)
    cout << "3)" << (*sp3).n << endl;  //输出 2
    sp1.reset();          //sp1放弃托管 A(2)
    if( !sp1 )
            cout << "4)sp1 is null" << endl; //会输出
    A * q = new A(3);
    sp1.reset(q); // sp1托管q
    cout << "5)" << sp1->n << endl;  //输出 3
    shared_ptr<A> sp4(sp1); //sp4托管A(3)
    shared_ptr<A> sp5;
    //sp5.reset(q); 不妥，会导致程序出错
    sp1.reset();          //sp1放弃托管 A(3)
    cout << "before end main" <<endl;
    sp4.reset();          //sp1放弃托管 A(3)
    cout << "end main" << endl;
    return 0; //程序结束，会delete 掉A(2)
}
```

输出结果：
1)2,2
2)2
3)2
4)sp1 is null
5)3
before end main
3 destructor
end main
2 destructor

```
#include <iostream>
#include <memory>
using namespace std;
struct A {
        ~A() { cout << "~A" << endl; }
};
int main()
{
        A * p = new A;
        shared_ptr<A> ptr(p);
        shared_ptr<A> ptr2;
        ptr2.reset(p); //并不增加ptr中对p的托管计数
        cout << "end" << endl;
        return 0;
}
```

输出结果：
end
~A
~A
之后程序崩溃，
因p被delete两次

空指针nullptr

```
#include <memory>
#include <iostream>
using namespace std;
int main()   {
        int* p1 = NULL;
        int* p2 = nullptr;
        shared_ptr<double> p3 = nullptr;
        if(p1 == p2)
                cout << "equal 1" <<endl;
        if( p3 == nullptr)
                cout << "equal 2" <<endl;
        if( p3 == p2) ; // error
        if( p3 == NULL)
                cout << "equal 4" <<endl;
        bool b = nullptr; // b = false
        int i = nullptr; //error,nullptr不能自动转换成整型
        return 0;
}
```

去掉出错的语句后输出：

equal 1

equal 2

equal 4

基于范围的for循环

```
#include <iostream>
#include <vector>
using namespace std;
struct A { int n;       A(int i):n(i) {         } };
int main()  {
        int ary[] = {1,2,3,4,5};
        for(int & e: ary)
                e*= 10;
        for(int e : ary)
                cout << e << ",";
        cout << endl;
        vector<A> st(ary,ary+5);
        for(  auto & it: st)
                it.n *= 10;
        for(  A it: st)
                cout << it.n << ",";
        return 0;
}
```

输出：
10,20,30,40,50,
100,200,300,400,500,

右值引用和move语义

右值：一般来说，不能取地址的表达式，就是右值，
能取地址的，就是左值

```
class A { };
A & r = A(); // error , A()是无名变量，是右值
A && r = A(); //ok, r 是右值引用
```

主要目的是提高程序运行的效率，减少需要进行深拷贝的对象进行深拷贝的次数。

参考

http://amazingjxq.com/2012/06/06/%E8%AF%91%E8%AF%A6%E8%A7%A3c%E5%8F%B3%E5%80%BC%E5%BC%95%E7%94%A8/

http://www.cnblogs.com/soaliap/archive/2012/11/19/2777131.html

```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;
class String
{
public:
        char * str;
        String():str(new char[1]) { str[0] = 0;}
        String(const char * s) {
                str = new char[strlen(s)+1];
                strcpy(str,s);
        }
        String(const String & s) {
                cout << "copy constructor called" << endl;
                str = new char[strlen(s.str)+1];
                strcpy(str,s.str);
        }
```

```
String & operator=(const String & s) {
        cout << "copy operator= called" << endl;
        if( str != s.str) {
                delete [] str;
                str = new char[strlen(s.str)+1];
                strcpy(str,s.str);
        }
        return * this;
}
   // move constructor
String(String && s):str(s.str) {
        cout << "move constructor called"<<endl;
        s.str = new char[1];
        s.str[0] = 0;
}
```

```
    // move assigment
        String & operator = (String &&s) {
            cout << "move operator= called"<<endl;
            if (str!= s.str) {
                    str = s.str;
                    s.str = new char[1];
                    s.str[0] = 0;
            }
            return *this;
        }
        ~String() { delete [] str; }
};
template <class T>
void MoveSwap(T& a, T& b)   {
    T tmp(move(a)); // std::move(a)为右值，这里会调用move constructor
    a = move(b);    // move(b)为右值，因此这里会调用move assigment
    b = move(tmp);  // move(tmp)为右值，因此这里会调用move assigment
}
```

```
int main()
{
        //String & r = String("this"); // error
        String s;
        s = String("this");
        cout << "****" << endl;
        cout << s.str << endl;
        String s1 = "hello",s2 = "world";
        MoveSwap(s1,s2);
        cout << s2.str << endl;
        return 0;
}
```

输出：

```
move operator= called
****
this
move constructor called
move operator= called
move operator= called
hello
```

In-Video Quiz

1. 下面的变量x是什么类型的（假设头文件都已经包含）？

```
template <class T1, class T2>
auto add(T1 x, T2 y) -> decltype(x + y)  {
    return x+y;
}
int main(){
                auto x = add( string("hello"),"world");
                return 0;
}
```

A)char * B)string C)int D)拜托，上面程序有语法错误好不好

2. struct A { int n; }; shared_ptr<A> p(new A());
则以下哪个表达式是没有定义的？
A)! p B)p->n; C)*p D)++p;

3. 以下哪段程序没有编译错误？
A) string & r = string("this");
B) string && r = string("this");
C) string s; string && r = s;
D) string s; string & r = s; string && rr = r;

In-Video Quiz

1. 下面的变量x是什么类型的（假设头文件都已经包含）？

```
template <class T1, class T2>
auto add(T1 x, T2 y) -> decltype(x + y)  {
    return x+y;
}
int main(){
                auto x = add( string("hello"),"world");
                return 0;
}
```

A)char * B)string C)int D)拜托，上面程序有语法错误好不好

#B

2. struct A { int n; }; shared_ptr<A> p(new A());

则以下哪个表达式是没有定义的？

A)! p B)p->n; C)*p D)++p;

#D

3. 以下哪段程序没有编译错误？

A) string & r = string("this");

B) string && r = string("this");

C) string s; string && r = s;

D) string s; string & r = s; string && rr = r;

#B