liufei_learning--脚踏实地，戒骄戒躁！

每天叫醒自己的是闹钟，而让自己起床的是梦想！

C++命名空间

分类： C/C++基础 2010-03-18 08:50 8269人阅读 评论(7) 收藏 举报

c++ fun header 语言 string c

一、 为什么需要命名空间（问题提出）

命名空间是ANSIC++引入的可以由用户命名的作用域，用来处理程序中 常见的同名冲突。

在C语言中定义了3个层次的作用域，即文件(编译单元)、函数和复合语句。C++又引入了类作用域，类是出现在文件内的。在不同的作用域中可以定义相同名字的变量，互不干扰，系统能够区别它们。

1、全局变量的作用域是整个程序，在同一作用域中不应有两个或多个同名的实体(enuty)，包括变量、函数和类等。

例： 如果在文件中定义了两个类，在这两个类中可以有同名的函数。在引用时，为了区别，应该加上类名作为限定：

```
class A      //声明A类
{ public:
    void funl(); //声明A类中的funl函数
private:
    int i; };
void A::funl() //定义A类中的funl函数
{.....}

class B //声明B类
{ public:
    void funl(); //B类中也有funl函数
    void fun2(); };
void B::funl() //定义B类中的funl函数
{ .....}
```

这样不会发生混淆。

在 文件中可以定义全局变量(global variable)，它的作用域是整个程序。如果在文件A中定义了一个变量a int a=3；在文件B中可以再定义一个变量a int a=5;

在分别对文件A和文件B进行编译时不会有问题。但是，如果一个程序包括文件A和文件B，那么在进行连接时，会报告出错，因为在同一个程序中有两个同名的变量，认为是对变量的重复定义。

可 以通过extern声明同一程序中的两个文件中的同名变量是同一个变量。如果在文件B中有以下声明：

```
extem int a;
```

表示文件B中的变量a是在其他文件中已定义的变量。由于有此声明，在程序编译和连接后，文件A的变量a的作用域扩展到了文件B。如果在文件B中不再对a赋值，则在文件B中用以下语句输出的是文件A中变量a的值： cout<<a； **//得到a的值为3**

2、程序中就会出现名字冲突。

我的大学--C++学习笔记 (17977)
《论语》全文 (11782)
C++命名空间 (8269)
C++学习笔记(一)--基础知 (5157)
int main(int argc, char* a (5155)
诸葛亮《诫子书》 (4721)
C51编译警告"*** WARN (4502)
VS2008错误：error PRJ (4374)
Picture control用法 (3816)
常见文件打开方式 (3608)

最新评论

汇编一日一学(9)--响铃N次
DongKhan: STACKS SEGMENT dw 128 dup(0)STACKS ENDS这段代码删...

C++笔记：面向对象编程基础
rierrt: 赞！好好学习下

C++笔记：复制控制
jiangjian0206: C++博大精深

C++笔记：Const
liufei_learning: @xwhbwas2008:看了你的问题，是问为什么返回值类型是const char对吧。因为cons...

C++笔记：Const
徐香芋: 楼主 可以来这里看看这个const的问题不 http://bbs.csdn.net/topics/39...

C++笔记：构造函数
明哥之家: 路过。

数据结构基础-Hash Table详解
java-er: 信息很全面

C++命名空间
piaorouge: 很透彻，赞一个。另外，header1.h和header2.h中的using namespace st...

C51编译警告"*** WARNING L1:
Ajoke: 谢谢博主，我为这问题困了两天，看来有必要再看看C语言基础了，以前没学好。

VS2010 -- fatal error C1189: #er
镁铝是神马: 你好，我改了工程中的stdafx.h，就是把你的文章里的那段代码粘贴上去了，重新build，但是还是...

在简单的程序设计中，只要人们小心注意，可以争取不发生错误。但是，一个大型的应用软件，往往不是由一个人独立完成的，而是由若干人合作完成的，不同的人分别完成不同的部分，最后组合成一个完整的程序。假如不同的人分别定义了类，放在不同的头文件中，在主文件(包含主函数的文件)需要用这些类时，就用#include命令行将这些头文件包含进来。由于各头文件是由不同的人设计的，有可能在不同的头文件中用了相同的名字来命名所定义的类或函数。

例4 名字冲突

程序员甲在头文件headerl．h中定义了类 Student和函数fun。

```
// 例4中的头文件header1(头文件1，没其文件名为cc8-4-h1.h)
#include <string>
#include <cmath>
using namespace std;
class Student //声明Student类
 { public:
   Student(int n,string nam,int a)
    { num=n;name=nam;age=a;}
   void get_data();
 private:
   int num;
   string name;
   int age; };
void Student::get_data() //成员函数定义
{ cout<<num<<" "<<name<<" "<<age<<endl; }
double fun(double a,double b)//定义全局函数(即外部函数)
{ return sqrt(a+b);}
```

在 main函数所在的文件中包含头文件headerl.h：

```
#include <iostream>
using namespace std;
#include "header1.h" //注意要用双引号，因为文件一般是放在用用户目录中的
int main()
{  Student stud1(101,"Wang",18); //定义类对象studl
  stud1.get_data();
  cout<<fun(5,3)<<endl;
  return 0; }
```

程序 能正常运行，输出为

```
101 Wang 18
2.82843
```

如果程序员乙写了头文件header2．h，在其中除了定义其他类以外，还定义了类Student和函数fun，但其内容与头文件headerl．h中的 Student和函数fun有所不同。

```
// 例4中的头文件header2
#include <string>
#include <cmath>
using namespace std;
class Student //声明Student类
{ public:
   Student(int n,string nam,char s)//参数与headerl中的student不同
   { num=n;name=nam;sex=s;}
   void get_data();
   private:
   int num;
   string name;
   char sex; };//此项与headerl不同

void Student::get_data()//成员函数定义
{ cout<<num<<" "<<name<<" "<<sex<<endl; }
```

```
double fun(double a,double b) //定义全局函数
  { return sqrt(a-b);} //返回值与headerl中的fun函数不同
  //头文件2中可能还有其他内容
```

假如主程序员在其程序中要用到headerl．h中的Student和函数fun，因而在程序中包含了头文件headerl．h，同时要用到头文件 header2．h中的一些内容(但对header2．h中包含与headerl.h中的Student类和fun函数同名而内容不同的类和函数并不知情，因为在一个头文件中往往包含许多不同的信息，而使用者往往只关心自己所需要的部分，而不注意其他内容)，因而在程序中又包含了头文件 header2．h。如果主文件(包含主函数的文件)如下：

```
#include <iostream>
using namespace std;
#include "header1.h"//包含头文件l
#include "header2.h"//包含头文件2
int main()
{ Student stud1(101,"Wang",18);
stud1.get_data();
cout<<fun(5,3)<<endl;
return 0; }
```

这时程序编译就会出错。因为在预编译后，头文件中的内容取代了对应的#include命令行，这样就在同一个程序文件中出现了两个Student类和两个 fun函数，显然是重复定义，这就是名字冲突，即在同一个作用域中有两个或多个同名的实体。

3、全局命名空间污染(global namespace pollution)。

在程序中还往往需要引用一些库(包括C++编译系统提供的库、由软件开发商提供的库或者用户自己开发的库)，为此需要包含有关的头文件。如果在这些库中包含有与程序的全局实体同名的实体，或者不同的库中有相同的实体名，则在编译时就会出现名字冲突。

为了避免这类问题的出现，人们提出了许多方法，例如：将实体的名字写得长—些(包含十几个或几十个字母和字符)；把名字起得特殊一些，包括一些特殊的字符；由编译系统提供的内部全局标识符都用下划线作为前缀，如_complex()，以避免与用户命名的实体同名；由软件开发商提供的实体的名字用特定的字符作为前缀。但是这样的效果并不理想，而且增加了阅读程序的难度，可读性降低了。

C 语言和早期的C++语言没有提供有效的机制来解决这个问题，没有使库的提供者能够建立自己的命名空间的工具。人们希望ANSI C++标准能够解决这个问题，提供—种机制、一种工具，使由库的设计者命名的全局标识符能够和程序的全局实体名以及其他库的全局标识符区别开来。

二、 什么是命名空间（解 决方案）

命名空间：实际上就是一个由程序设计者命名的内存区域，程序设计者可以根据需要指定一些有名字的空间域，把一些全局实体分别放在各个命名空间中，从而与其他全局实体分隔开来。

如：

```
namespace ns1 //指定命名中间nsl
  { int a;
  double b; }
```

namespace 是定义命名空间所必须写的**关键字**，nsl 是用户自己指定的**命名空间的名字**(可 以用任意的合法标识符，这里用ns1是因为ns是namespace的缩写，含义请楚)，在花括号内是声明块，在其中声明的实体称为**命名空间成员(namespace member)。**现在命名空间成员包括变量a和b，注意a和b仍然是全局变量，仅仅是把它们隐藏在指定的命名空间中而已。如果在程序中要使用变量a和b，必须加上命名空间名和作用域分辨符“::”，如nsl::a，nsl::b。这种用法称为**命名空间限定(qualified)**，这些名字(如nsl::a)称为被限定名 (qualified name)。C++中命名空间的作用类似于操作系统中的目录和文件的关系，由于文件很多，不便管理，而且容易重名，于是人们设立若干子目录，把文件分别放到不同的子目录中，不同子目录中的文件可以同名。调用文件时应指出文件路径。

命名空间的作用：是建立一些互相分隔的作用域，把一些全局实体分隔开来。以免产生老点名叫李相国时，3个人都站起来应答，这就是名字冲突，因为他们无法辨别老师想叫的是哪一个李相国，同名者无法互相区分。为了避免同名混淆，学校把3个同名的学生分在3个班。这样，在小班点名叫李相国时，只会有一个人应答。也就是说，在该班的范围(即班作用域)内名字是惟一的。如果在全校集合时校长点名，需要在全校范围内找这个学生，就需要考虑作用域问题。如果校长叫李相国，全校学生中又会有3人一齐喊“到”，因为在同一作用域中存在3个同名学生。为了在全校范围内区分这3名学生，校长必须在名字前加上班号，如高三甲班的李相国，或高三乙班的李相国，即加上班名限定。这样就不致产生混淆。

可以根据需要设置许多个命名空间，每个命名空间名代表一个不同的命名空间域，不同的命名空间不能同名。这样，

可以把不同的库中的实体放到不同的命名空间中，或者说，用不同的命名空间把不同的实体隐蔽起来。过去我们用的全局变量可以理解为全局命名空间，独立于所有有名的命名空间之外，它是不需要用 namespace声明的，实际上是由系统隐式声明的，存在于每个程序之中。

在声明一个命名空间时，花括号内不仅可以包括变量，而且还可以包括以下类型：

·变量(可以带有初始化)；

·常量；

·数(可以是定义或声明)；

·结构体；

·类；

·模板；

·命名空间(在一个命名空间中又定义一个命名空间，即嵌套的命名空间)。

例如

```
namespace nsl
  { const int RATE=0.08； //常量
  doublepay；        //变量
  doubletax()        //函数
    {return a*RATE；}
  namespacens2       //嵌套的命名空间
    {int age；}
  }
```

如果想输出命名空间nsl中成员的数据，可以采用下面的方法：

```
cout<<nsl::RATE<<endl；
cout<<nsl::pay<<endl；
cout<<nsl::tax()<<endl；
cout<<nsl::ns2::age<<endl； //需要指定外层的和内层的命名中间名
```

可以看到命名空间的声明方法和使用方法与类差不多。但它们之间有一点差别：在声明类时在右花括号的后面有一分号，而在定义命名空间时，花括号的后面没有分 号。

三、 使用命名空间解决名字冲突（使用指南）

有了以上的基础后，就可以利用命名空间来解决名字冲突问题。现在，对例4程序进行修改，使之能正确运行。

例5 利用命名空间来解决例4程序名字冲突问题。

修改两个头文件，把在头文件中声明的类分别放在两个不同的命名空间中。

```
//例8.5中的头文件1，文件名为header1.h
using namespace std;
#include <string>
#include <cmath>
namespace ns1 //声明命名空间ns1
{ class Student //在命名空间nsl内声明Student类
{ public:
Student(int n,string nam,int a)
{ num=n;name=nam;age=a;}
void get_data();
private:
int num;
string name;
int age; };
void Student::get_data() //定义成员函数
  { cout<<num<<" "<<name<<" "<<age<<endl; }

double fun(double a,double b) //在命名空间n引内定义fun函数
{ return sqrt(a+b);}
}
//例 8.5中的头文件2，文件名为header2.h
#include <string>
```

```
#include <cmath>
namespace ns2 //声明命名空间ns2
{ class Student
{ public:
Student(int n,string nam,char s)
{ num=n;name=nam;sex=s;}
void get_data();
private:
int num;
string name;
char sex; };

void Student::get_data()
{ cout<<num<<" "<<name<<" "<<sex<<endl; }

double fun(double a,double b)
{ return sqrt(a-b);}
}
//main file
#include <iostream>
#include "header1.h" //包含头文件l
#include "header2.h" //包含头文件2
int main()
{ ns1::Student stud1(101,"Wang",18);//用命名空间nsl中声明的Student类定义studt
stud1.get_data(); //不要写成ns1::studl.get_data()；
cout<<Ns1::fun(5,3)<<endl; //调用命名空间ns1中的fun函数
ns2::Student stud2(102,"Li",'f'); //用命名空间ns2中声明的 Student类定义stud2
stud2.get_data();
cout<<ns2::fun(5,3)<<endl; //调用命名空间nsl，中的fun函数
return 0; }
```

解决本题的关键是建立了两个命名空间nsl和ns2，将原来在两个头文件中声叫的类分别放在命名空间nsl和ns2中。注意：在头文件中，不要把#include命令放在命名空间中，在上一小节的叙述中可以知道，**命名空间中的内容不包括命令行，否则编译会出错。**

分析例4程序出错的原因是：在两个头文件中有相同的类名Student和相同的函数名fun，在把它们包含在主文件中时，就产生名字冲突，存在重复定义。编译系统无法辨别用哪一个头文件中的Student来定义对象studl。现在两个Student和fun分别放在不同的命名空间中，各自有其作用域，互不相干。由于作用域不相同，不会产：生名字冲突。正如同在两个不同的类中可以有同名的变量和函数而不会产生冲突一样。

在定义对象时用ns1::Student(命名空间nsl中的Student)来定义studl，用ns2：：Student(命名空间ns2中的 Student)来定义stud2。显然，nsl::Student和ns2::Student是两个不同的类，不会产生混淆。同样，在调用fun函数时也需要用命名空间名ns]或ns2加以限定。ns1::fun()和ns2::fun()是两个不同的函数。注意：对象studl是用 nsl：：Student定义的，但对象studl并不在命名空间nsl中。studl的作用域为main函数范围内。在调用对象studl的成员函数 get_data时，应写成studl．get_data()，而不应写成nsl::studl.get_data()。

程序 能顺利通过编译，并得到以下运行结果：

101 Wang l9 (对象studl中的数据)

2.82843 (／5+3的值)

102 Li f (对象studg中的数据)

1.41421 (／5-2的值)

四、 使用命名空间成员的方法

从上面的介绍可以知道，在引用命名空间成员时，要用命名空间名和作用域分辨符对命名空间成员进行限定，以区别不同的命名空间中的同名标识符。即:

命名空间名：：命名空间成员名

这种方法是有效的，能保证所引用的实体有惟一的名字。但是如果命名空间名字比较长，尤其在有命名空间嵌套的情况下，为引用一个实体，需要写很长的名字。在一个程序中可能要多次引用命名空间成员，就会感到很不方便。

1、使用命名空间别名

可以为命名空间起一个别名(namespace alias)，用来代替较长的命名空间名。**如**

```
namespace Television //声明命名空间，名为Television
{ ... }
```

可以用一个较短而易记的别名代替它。如：

namespace TV=Television； //别名TV与原名Television等价

也可以说，别名TV指向原名Television，在原来出现Television的位置都可以无条件地用TV来代替。

2、使用using命名空间成员名

using后面的命名空间成员名必须是由命名空间限定的名字。**例如：**

```
using nsl::Student；
```

以上语句声明：在本作用域(using语句所在的作用域)中会用到命名空间ns1中的成员Student，在本作用域中如果使用该命名空间成员时，不必再用命名空间限定。例如在用上面的using声明后，在其后程序中出现的Student就是隐含地指nsl::Student。

using声明的有效范围是从using语句开始到using所在的作用域结束。如果在以上的using语句之后有以下语句：

Student studl(101,"Wang",18)； //此处的Student相当于ns1::Student

上面的语句相当于

nsl::Student studl(101，"Wang"，18)；

又如

```
using nsl::fun； //声明其后出现的fun是属于命名空间nsl中的fun
cout<<fun(5，3)<<endl；//此处处的fun函数相当于nsl::fun(5，3)
```

显然，这可以避免在每一次引用命名空间成员时都用命名空间限定，使得引用命名空间成员变得方便易用。

但是要注意：在同一作用域中用using声明的不同命名空间的成员中不能有同名的成员。例如：

```
usmgnsl::Student； //声明其后出现的Student是命名空间nsl中的Student
usmgns2::Student； //声 明其后出现的Student是命名空间ns2小的Student
Student stud1； //请问此处的Student是哪个命名中间中的Student?
```

产生了二义性，编译出错。

3、使用using namespace命名空间名

用上面介绍的using命名空间成员名，一次只能声明一个命名空间成员，如果在一个命名空间中定义了10个实体，就需要使用10次using命名空间成员名。能否在程序中用一个语句就能一次声明一个命名空间中的全部成员呢?

C++提供了using namespace语句来实现这一目的。using namespace语句的一般格式为

```
using namespace 命名空间名；
例如
using nanlespace nsl；
```

声明了在本作用域中要用到命名空间nsl中的成员，在使用该命名空间的任何成员时都不必用命名空间限定。如果在作了上面的声明后有以下语句：

```
Student studl(101，”Wang”，18)；//Student隐含指命名中间nsl中的Student
cout<<fun(5，3)<<endl；//这里的fun函数是命名中间 nsl中的fun函数
```

在用usmgnamespace声明的作用域中，命名空间nsl的成员就好像在全局域声明的一样。因此可以不必用命名空间限定。显然这样的处理对写程序比较方便。但是如果同时用usingnamespace声明多个命名空间时，往往容易出错。例5中的main函数如果用下面程序段代替，就会出错。

```
int main()
{ using namespace nsl；//声明nsl中的成员在本作用域中可用
using namespace ns2；//声明ns2中的成员在本作用域中可用
Student studl(101，”Wang"，18)；
studl.8ct_data()；
cout<<fun(5,3)<<endl；
Student stud2(102，"Li"，'r')；
stud2.get_data()；
coutt<<fun(5，3)<<endl；
return O； }
```

因为在同一作用域中同时引入了两个命名空间nsl和ns2，其中有同名的类和函数。在出现Student时，无法判定是哪个命名空间中的 Student，出现二义性，编译出错。因此只有在使用命名空间数量很少，以及确保这些命名空间中没有同名成员时才用using namespace语句。

五、 无名的命名空间

以上介绍的是有名字的命名空间，C++还允许使用没有名字的命名空间，如在**文件A中声明了以下的无名命名**

空间：

```
namespace //命名空间没有名字
{ void fun( ) //定 义命名空间成员
{ cout<<"OK．"<<endl；}
}
```

由于命名空间没有名字，在其他文件中显然无法引用，它只在本文件的作用域内有效。无名命名空间的成员fun函数的作用域为文件A(确切地说，是从声明无名命名空间的位置开始到文件A结束)。在文件A中使用无名命名空间的成员，不必(也无法)用命名空间名限定。

如果 在文件A中有以下语句：

fun()；

则执行无名命名空间中的成员fun函数，输出”OK．”。

在本程序中的其他文件中也无法使用该fun函数，也就是把fun函数的作用域限制在本文件范围中。可以联想到：在C浯言中可以用static声明一个函数，其作用也是使该函数的作用域限于本文件。C++保留了用static声明函数的用法，同时提供了用无名命名空间来实现这一功能。随着越来越多的C++ 编译系统实现了ANSI C++建议的命名空间的机制，相信使用无名命名空间成员的方法将会取代以前习惯用的对全局变量的静态声明。

六、标准命名空间std

为了解决C++标准库中的标识符与程序中的全局标识符之间以及不同库中的标识符之间的同名冲突，应该将不同库的标识符在不同的命名空间中定义(或声明)。标准C++库的所有的标识符都是在一个名为std的命名空间中定义的，或者说标准头文件(如iostream)中函数、类、对象和类模板是在命名空间 std中定义的。std是standard(标准)的缩写，表示这是存放标准库的有关内容的命名空间，含义请楚，不必死记。

这样，在程序中用到C++标准库时，需要使用std作为限定。如

std::cout<<"OK．"<<endl； //声明cout是在 命名空间std中定义的流对象

在有的C++书中可以看到这样的用法。但是在每个cout，cm以及其他在std中定义的标识符前面都用命名空间std作为限定，显然是很不方便的。在大多数的C++程序中常用usmgnamespace语句对命名空间std进行声明，这样可以不必对每个命名空间成员一进行处理，在文件的开头加入以下 using namespace声明：

```
using namespace std；
```

这样，在std中定义和声明的所有标识符在本文件中都可以作为全局量来使用。但是应当绝对保证在程序中不出现与命名空间std的成员同名的标识符，例如在程序中不能再定义一个名为cout的对象。由于在命名空间std中定义的实体实在太多，有时程序设计人员也弄不请哪些标识符已在命名空间std中定义过，为减少出错机会，有的专业人员喜欢用若干个"using命名空间成员”声明来代替“using namespace命名空间”声明，如

```
using Std::string；
using Std::cout；
using Std::cin；
```

等。为了减少在每一个程序中都要重复书写以亡的using声明，程序开发者往往把编写应用程序时经常会用到的命名空间std成员的usmg声明组成一个头文件，然后在程序中包含此头文件即可。

如果阅读了多种介绍C++的书，可能会发现有的书的程序中有using namespace语句，有的则没有。有的读者会提出：究竟应该有还是应该没有?应当说：用标准的C++编程，是应该对命名空间std的成员进行声明或限定的(可以采取前面介绍过的任一种方法)。但是目前所用的C++库大多是几年前开发的，当时并没有命名空间，库中的有关内容也没有放在std命名空间中，因而在程序中不必对std进行声明。

七、 使用早期的函数库

C语言程序中各种功能基本上都是由函数来实现的，在C语言的发展过程中建立了功能丰富的函数库，C++从C语言继承了这份宝贵的财富。在C++程序中可以使用C语言的函数库。

如果要用函数库中的函数，就必须在程序文件中包含有关的头文件，在不同的头文件中，包含了不同的函数的声明。

在C++中使用这些 头文件有两种方法。

1、用C语言的传统方法

头文件名包括后缀．h，如stdio.h，math.h等。由于C语言没有命名空间，头文件并不存放在命名空间中，因此在C++程序文件中如果用到带后缀．h的头文件时，不必用命名空间。只需在文件中包含所用的头文件即可。**如**

```
#include<math．h>
```

2、用C++的新方法

C++标准要求系统提供的头文件不包括后缀．h，例如iostream、string。为了表示与C 语言的头文件有联系又有区别，C++所用的头文件名是在C语言的相应的头文件名(但不包括后缀．h)之前加一字母c。例如，C语言中有关输入与输出的头文件名为stdio．h在C++中相应的头文件名为cstdio。C语言中的头文件math.h，在C++中相应的头文什名为cmath。C语言中的头文件 string．h在C++中相应的头文件名为cstring。注意在C++中，头文件cstnng和头文件strmg不是同一个文件。前者提供C语言中对字符串处理的有关函数(如strcmp，ctrcpy)的声明，后者提供C++中对字符串处理的新功能。

此外，由于这些函数都是在命名空间std中声明的，因此在程序中要对命名空间std作声明。如：

```
#include<cstdio>
#include<cmath>
using namespace std;
```

目前所用的大多数C++编译系统既保留了c的用法，又提供了C++的新方法。下面两种用法等价，可以任选。

C传 统方法 C++新方法

#include<stdio.h> #include<cstdio>

#include<math.h> #include<cmath>

#include<string.h> #include<cstring>

using namespace std;

可以使用传统的c方法，但应当提倡使用C++的新方法。

源文档 <http://hi.baidu.com/rainysky_2006/blog/item/a490e01fc3de7964f724e4d1.html>

更多 3

上一篇 面试技巧:16个经典面试问题回答思路

下一篇 stdafx.h头文件的作用

开发 应用程序 程序开发

!入 android开发之设置Edittext密码的方法

【iOS SOAP】基于第三方开源项目：wsdl2objc

专业课133分过来人谈计算机专业考研各科目复习方

Linux环境安装ACE 6.1.0

l) 一步步进行 LUA的OOD封装 (一)

查看评论

7楼 piaorouge 2013-12-02 14:07发表

很透彻，赞一个。另外，header1.h和header2.h中的using namespace std去掉是不是好一些，有命名空间污染。

6楼 guorongshan 2013-04-15 15:10发表

赞一个，讲解的很透。

5楼 BYD123 2012-10-30 10:34发表

不错。

4楼 E_l_e_g_y 2012-08-29 15:09发表

讲的比较细致，解决了疑惑，谢谢

3楼 alice_yangmin 2012-07-11 09:58发表

不错，挺细的

2楼 木瓜 2012-06-29 23:50发表

不错，比较容易懂！

1楼 zhulintingfen 2012-03-18 10:43发表

奇文 赞一个

您还没有登录,请[登录]或[注册]