目录

[iostream 与iostream.h区别 2](#_Toc336075836)

[C和C++的bool类型 3](#_Toc336075837)

[静态变量初始化 4](#_Toc336075838)

[STL 4](#_Toc336075839)

[C++ static、const和static const 以及它们的初始化 5](#_Toc336075840)

[【转】VS下怎么写纯C程序 10](#_Toc336075841)

[虚函数的注意事项 12](#_Toc336075842)

[C++中typedef struct与struct的区别 14](#_Toc336075843)

[回调函数 25](#_Toc336075844)

[回调函数实现的机制是 25](#_Toc336075845)

[编辑本段为什么要使用回调函数 25](#_Toc336075846)

[编辑本段简单的回调函数实现 26](#_Toc336075847)

[代码实现 26](#_Toc336075848)

[调用约定 28](#_Toc336075849)

[运算符优先级 29](#_Toc336075850)

[编辑本段语言操作符 29](#_Toc336075851)

[编辑本段三个优先级 30](#_Toc336075852)

[编辑本段算术运算符 30](#_Toc336075853)

[编辑本段C++运算符优先级 31](#_Toc336075854)

[C++中的vector使用范例 34](#_Toc336075855)

[C++中cin.get和cin.getline()的区别 37](#_Toc336075856)

[c++强制转化 39](#_Toc336075857)

[TCP/IP协议 41](#_Toc336075858)

[层次概述 42](#_Toc336075859)

[网络接口层 42](#_Toc336075860)

[网络层 42](#_Toc336075861)

[传输层 43](#_Toc336075862)

[应用层 43](#_Toc336075863)

[数据格式 44](#_Toc336075864)

[产生背景 44](#_Toc336075865)

[开发过程 45](#_Toc336075866)

[运作机制 46](#_Toc336075867)

[1.IP 46](#_Toc336075868)

[2.TCP 46](#_Toc336075869)

[3.UDP 47](#_Toc336075870)

[4.ICMP 47](#_Toc336075871)

[通讯端口 47](#_Toc336075872)

[IP地址 48](#_Toc336075873)

[地址分类 48](#_Toc336075874)

[网关地址 50](#_Toc336075875)

[注意事项 50](#_Toc336075876)

[子网划分 50](#_Toc336075877)

[子网掩码 50](#_Toc336075878)

[子网作用 52](#_Toc336075879)

[概念区分 52](#_Toc336075880)

[协议阐述IP申请的术语 52](#_Toc336075881)

[子网划分 53](#_Toc336075882)

[协议测试 53](#_Toc336075883)

[主要特点 53](#_Toc336075884)

[协议优势 53](#_Toc336075885)

[协议重置 54](#_Toc336075886)

[版本更新 54](#_Toc336075887)

[IPV4 54](#_Toc336075888)

[IPv6 55](#_Toc336075889)

[以太网，万维网，因特网，互联网 55](#_Toc336075890)

[中缀表达式转后缀表达式 56](#_Toc336075891)

[二叉树的深度优先遍历、广度优先遍历和非递归遍历 57](#_Toc336075892)

# [iostream 与iostream.h区别](http://blog.csdn.net/wzwind/article/details/1686217)

C++的标准类库被修订了两次，有两个标准 C92和C99,这两个库现在都在并行使用，用 .h 包含的是c92 ，不带 .h 的是c99的头文件，对于普通用户来说这两者没有什么区别，区别是在内部函数的具体实现上。旧的C++头文件是官方明确反对使用的，但旧的C头文件则没有（以保持对C的兼容性）。据说从 Visual C++ .NET 2003 开始，移除了旧的 iostream 库。其实编译器制造商不会停止对客户现有软件提供支持，所以在可以预计的将来，旧的C++头文件还会嚣张一段时间。如果能明白字符串头文件的使用，举一反三，其他的也差不多会用了:

<string.h>是旧的C/C++头文件，对应的是基于char\*的字符串处理函数；

<string>是包装了std的C++头文件，对应的是新的strng类；

<cstring>是对应旧的C头文件的std版本。

如果编译器都同时支持< iostream >和< iostream.h >，那使用#include < iostream >，得到的是置于名字空间std下的iostream库的元素；如果使用#include < iostream.h >，得到的是置于全局空间的同样的元素。在全局空间获取元素会导致名字冲突，而设计名字空间的初衷正是用来避免这种名字冲突的发生。

源文档 <<http://blog.csdn.net/wzwind/article/details/1686217>>

2010-10-10 21:26

# C和C++的bool类型

起初的C语言没有bool类型，通常用int来当作布尔类型来使用，五花八门的

C标准委员会终于在C99标准中引入了标准的定义:

\_Bool类型，值域为0和1。

\_Bool类型不能用signed, unsigned, short, long等修饰。

标准没有规定\_Bool类型占用的大小。不过由于多数C编译器是和C++编译器一块供应

的，因此在实现了C99的编译器上，\_Bool类型的大小跟C++编译器的bool一样。

之所以用\_Bool这个怪异的名字，是因为怕和以前已经定义自己的布尔类型的源代码

冲突，而且上一版的C标准也规定了下划线开头跟一个大写字母或者又一个下划线的

标识符不是给用户用的，你如果用了而造成冲突那就管你自个儿倒霉吧。

不过\_Bool的确很难看，而且0和1作为其取值也不够清晰，为了让代码更清晰一些，

标准又规定了一个标准头文件，叫做stdbool.h，对其规定如下：

要定义一个bool宏，展开后为\_Bool

要定义一个true宏，展开后为1

要定义一个false宏，展开后为0

要定义一个宏常量，名字叫\_\_bool\_true\_false\_are\_defined，值为1，方便程序中用

条件预处理检测之。

有了这个头文件，就可以像C++中的bool一样在C中使用bool了。

不过不幸的是，目前C99标准推出后一直受冷落，所以C99以后的VC，从VC7直到最新

的VS2005，都没实现C99。所幸的是stdbool.h很容易自己实现，因此自己定义一个拿

来用也不是什么难事。

C++中的bool类型

C++一开始也没有bool类型，后来引入了bool类型。

bool类型不能用signed, unsigned, short, long等修饰。

bool类型的值只能为false和true，bool值作整数运算的时候，false转为0，true转

为1。任何其他类型转为bool的时候，零转为false，非零转为true。

因此，下面的代码。

bool b = (bool)6;

int a = b;

b的值为true，a的值为1。

从其他类型转为bool类型会带来一定的效率损失，因此上面的第一句在某些编译器，

比如VC6商会产生警告，这么写就可以避免这样的警告：

bool b = (6 != 0);

通过内存操作故意让bool类型的左值取false和true以外的值是不厚道的。

bool在C++中可以参加函数的重载决议，这是C++中引入bool类型的一个重要原因。

C++也没有规定bool类型的宽度，只规定能放得下0和1两个值就可以，很多编译器选

择了一个字节，包括gcc和VC6以后的版本，VC5的bool类型的宽度是四个字节。

C++还规定对一个bool左值做++运算，其值为true，这一点仅为了兼容，实际代码中

不要使用。

布尔类型的引入也给C和C++的逻辑运算带来了变化，在引入布尔类型之前，C和C++的

逻辑运算的结果为整形，if，while等条件语句的条件也是整形；引入布尔类型之后

都变成了布尔类型。

源文档 <<http://hi.baidu.com/minyou_1314/item/68a70677a18c39500d0a071b>>

# 静态变量初始化

**Visual Studio .NET 2003**

此主题尚未评级 - [评价此主题](http://technet.microsoft.com/zh-cn/library/cc437200#feedback)

可以将静态局部变量的值初始化为其声明的一部分。如果将数组声明为 **Static**，则可以初始化它的秩（维数）、每个维度的长度和每个元素的值。

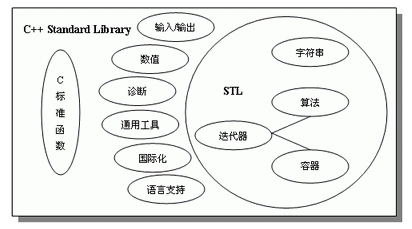
可以在多个过程中用相同的名称来声明静态变量。如果这样做了，Visual Basic 编译器会将每个这样的变量视为单独的元素。初始化这些变量中的某一个不会影响其他变量的值。这同样适用于使用一组重载来定义过程、并在每个重载中使用相同的名称声明静态变量的情况。

可以在类中（即在该类的过程内）声明静态局部变量。但是，**不能在结构中声明静态局部变量。**

静态变量初始化是由编译完成的。程序还没有运行时，这个值就要定下来,可以是类对象

# STL

STL = Standard Template Library，[标准模板库](http://baike.baidu.com/view/3097879.htm)，[惠普实验室](http://baike.baidu.com/view/2827626.htm)开发的一系列[软件](http://baike.baidu.com/view/37.htm)的统称。它是由Alexander Stepanov、Meng Lee和David R Musser在惠普实验室工作时所开发出来的。这可能是一个历史上最令人兴奋的工具的最无聊的术语。从根本上说，STL是一些“容器”的集合，这些“容器”有list,vector,set,map等，STL也是算法和其他一些组件的集合。这里的“容器”和算法的集合指的是世界上很多聪明人很多年的杰作。STL的目的是标准化组件，这样就不用重新开发，可以使用现成的组件。STL现在是C++的一部分，因此不用额外安装什么。



# [C++ static、const和static const 以及它们的初始化](http://blog.csdn.net/yjkwf/article/details/6067267)

2010-12-10 11:252454人阅读[评论](http://blog.csdn.net/yjkwf/article/details/6067267#comments)(0)[收藏](javascript:void(0);)[举报](http://blog.csdn.net/yjkwf/article/details/6067267#report)

const定义的常量在超出其作用域之后其空间会被释放，而static定义的静态常量在函数执行后不会释放其存储空间。

static表示的是静态的。类的静态成员函数、静态成员变量是和类相关的，而不是和类的具体对象相关的。即使没有具体对象，也能调用类的静态成员函数和成员变量。一般类的静态函数几乎就是一个全局函数，只不过它的作用域限于包含它的文件中。

在C++中，static静态成员变量不能在类的内部初始化。在类的内部只是声明，定义必须在类定义体的外部，通常在类的实现文件中初始化，如：double Account::Rate=2.25;static关键字只能用于类定义体内部的声明中，定义时不能标示为static

在C++中，const成员变量也不能在类定义处初始化，只能通过构造函数初始化列表进行，并且必须有构造函数。

const数据成员 只在某个对象生存期内是常量，而对于整个类而言却是可变的。因为类可以创建多个对象，不同的对象其const数据成员的值可以不同。所以不能在类的声明中初始化const数据成员，因为类的对象没被创建时，编译器不知道const数据成员的值是什么。

const数据成员的初始化只能在类的构造函数的初始化列表中进行。要想建立在整个类中都恒定的常量，应该用类中的枚举常量来实现，或者static cosnt。

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/yjkwf/article/details/6067267)

1. class Test
2. {
3. public:
4. Test():a(0){}
5. enum {size1=100,size2=200};
6. private:
7. const int a;//只能在构造函数初始化列表中初始化
8. static int b;//在类的实现文件中定义并初始化
9. const static int c;//与 static const int c;相同。
10. };
11. int Test::b=0;//static成员变量不能在构造函数初始化列表中初始化，因为它不属于某个对象。
12. cosnt int Test::c=0;//注意：给静态成员变量赋值时，不需要加static修饰符。但要加cosnt



cosnt成员函数主要目的是防止成员函数修改对象的内容。即const成员函数不能修改成员变量的值，但可以访问成员变量。当方法成员函数时，该函数只能是const成员函数。

static成员函数主要目的是作为类作用域的全局函数。不能访问类的非静态数据成员。类的静态成员函数没有this指针，这导致：1、不能直接存取类的非静态成员变量，调用非静态成员函数2、不能被声明为virtual

关于static、const、static cosnt、const static成员的初始化问题：

1、类里的const成员初始化：

在一个类里建立一个const时，不能给他初值

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/yjkwf/article/details/6067267)

1. class foo
2. {
3. public:
4. foo():i(100){}
5. private:
6. const int i=100;//error!!!
7. };
8. //或者通过这样的方式来进行初始化
9. foo::foo():i(100)
10. {}



2、类里的static成员初始化：

类中的static变量是属于类的，不属于某个对象，它在整个程序的运行过程中只有一个副本，因此不能在定义对象时 对变量进行初始化，就是不能用构造函数进行初始化，其正确的初始化方法是：

数据类型 类名::静态数据成员名=值；

**[c-sharp]** [view plaincopyprint?](http://blog.csdn.net/yjkwf/article/details/6067267)

1. class foo
2. {
3. public:
4. foo();
5. private:
6. static int i;
7. };
8. int foo::i=20;
9. 这表明：
10. 1、初始化在类体外进行，而前面不加static，以免与一般静态变量或对象相混淆
11. 2、初始化时不加该成员的访问权限控制符private、public等
12. 3、初始化时使用作用域运算符来表明它所属的类，因此，静态数据成员是类的成员而不是对象的成员。



3、类里的static cosnt 和 const static成员初始化

这两种写法的作用一样，为了便于记忆，在此值说明一种通用的初始化方法：

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/yjkwf/article/details/6067267)

1. class Test
2. {
3. public:
4. static const int mask1;
5. const static int mask2;
6. };
7. const Test::mask1=0xffff;
8. const Test::mask2=0xffff;
9. //它们的初始化没有区别，虽然一个是静态常量一个是常量静态。静态都将存储在全局变量区域，其实最后结果都一样。可能在不同编译器内，不同处理，但最后结果都一样。



这是一个完整的例子：

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/yjkwf/article/details/6067267)

1. #ifdef A\_H\_
2. #define A\_H\_
3. #include <iostream>
4. using namespace std;
5. class A
6. {
7. public:
8. A(int a);
9. static void print();//静态成员函数
10. private:
11. static int aa;//静态数据成员的声明
12. static const int count;//常量静态数据成员（可以在构造函数中初始化）
13. const int bb;//常量数据成员
14. };
15. int A::aa=0;//静态成员的定义+初始化
16. const int A::count=25;//静态常量成员定义+初始化
17. A::A(int a):bb(a)//常量成员的初始化
18. {
19. aa+=1;
20. }
21. void A::print()
22. {
23. cout<<"count="<<count<<endl;
24. cout<<"aa="<<aa<<endl;
25. }
26. #endif
27. void main()
28. {
29. A a(10);
30. A::print();//通过类访问静态成员函数
31. a.print();//通过对象访问静态成员函数
32. }

# 【转】VS下怎么写纯C程序

所谓的纯C程序，是指该程序（包括可执行程序或库）之源代码所用语法严格遵守C89标准，且所用之库函数，要么为C89标准所支持之标准库函数，要么为第三方以纯C程序所编写之库函数。  
上面是我对“纯C程序”做的一个非标准的、递归的定义。  
使 用Visual Studio的朋友们都知道，作为Visual Studio套件的Visual C++集成开发环境并没有提供C语言编译器，我们只能 使用C++编译器对C程序进行编译。尽管Eckel Bruce认为，用C++编译器来编译C程序可以发现程序中隐含的可能的错误（参见 《Think in C++》），但我坚持认为这不是一个好办法：这只会让大家写出那些运用了大量C++特性的C程序。这种程序是不可取的。 Bjnare Stroustrup也认为，C++应当是一门独立于C语言的、全新的程序设计语言；而我却觉得，C应该是独立于C++的、老树常青的语 言。因此，我们需要一个能够编写纯C程序的开发环境和编译器。  
Visual Studio .NET中的C++编译器为我们提供了这样功 能！Visual Studio .NET中的C++编译器不仅大大加强了对C++标准的支持，通过一些配置，还可以对纯C程序进行编译！在本文中我将对 其配制方法进行一个简要的介绍，希望能够对那些喜欢纯C程序设计的朋友们有所帮助。也希望初学C语言程序设计的朋友能够写出漂亮的C程序。  
集成开发环境的配置  
首先，在建立项目的时候，选择Win32控制台应用程序即可，但一定要选择“空项目”选项。这样就能防止Visual Studio向导为我们产生不必要的源文件（代码）和设置一些不必要的编译器选项。具体操作方法见如下图示：  
选择“文件”-〉“新建”-〉“项目”：  
在“新建项目”对话框中，在“项目类型”李表框中选择“Visual C++项目”，并在“模板”列表框中选择“Win32 控制项目”模板。别忘了在“名称”文本框中输入项目的名称，这里我起名为“**PureC**”：  
点击“确定”按钮后，将弹出“Win32 应用程序向导”对话框，别急着点“确定”：  
最初，在我看到这个对话框的时候，还真没注意过“应用程序设置”这个选项（可能是我太笨了罢-\_-b）。因此，我们要：  
点击“应用程序设置”选项卡，选中“附加选项”中的“空项目”：  
这 一步是问题的关键。如果我们没有选择“空项目”，Visual Studio .NET的向导会为我们添加一些支持用于支持Windows应用程序的源文 件（如“stdafx.h”、“stdafx.c”等），而且还会对编译器进行一些设置。尽管这些设置对于编写Windows API程序非常有用，但对 于我们编写纯C程序来讲，是非常不利的。因此，我们有必要选择“空项目”。  
点击“确定”按钮后，我们便得到了一个空的项目，观察一下“解决方案资源管理器”列表框，我们发现，已经没有了向导为我们产生的烦人的多余的源文件了。这时，我们需要手动地将我们的源文件添加到项目中。在这里我们添加一个新的源文件“**PureC**.c”。在“解决方案资源管理器”列表框中的“源文件”文件夹图表上单击右键，选择“添加”-〉“添加新项”：  
在弹出的“添加新项”对话框中，选择“C++ 文件”即可。并在“名称”文本框中输入源文件的名称。注意，必须在文件名中写明扩展名“.c”，否则IDE将自动为该文件添加“.cpp”扩展名：  
接下来，我们要设置一下项目的属性。在“解决方案资源管理器”的项目（这里是“**PureC**”）上单击右键，在弹出菜单中选择“属性”。将弹出“属性页”对话框。在左侧的树状视图中展开“C/C++”节点，选择“高级”节点；在右侧出现的列表框中，将“编译为”列表项改为“编译为 C 代码（/TC）”：  
点击“确定”按钮，OK，大功告成了。  
控制台选项的配置  
这个题目有点……那个……了。我的直观感觉就是用控制台命令行编译程序的人应该都是虾（就算不是大虾，也得是个小虾），对于命令行的选项应该是了解的很了。我也就不过多献丑了，简单而言，只要在命令行中添加/TC选项，就可以把一个源文件编译为纯C代码。  
测试  
经过这样一番调教，我们中有了一个能够编写纯C代码的环境了。空口无凭，我们得用一段程序验证一下。为我们的**PureC**.c添加如下一段代码：  
/\*  
\* Model: **PureC**.c  
\* Author: lover\_P  
\* Date: 2004-6-3  
\*  
\* Test the pure C envirment in VS.NET  
\*/  
struct A {  
int i;  
};  
int main() {  
A a;  
a.i = 10;  
}  
/\* end \*/  
这 段代码对于C++程序来说是完全正确的，完全可以通过编译且没有错误和警告。但这并不是“纯C程序”，因为在C标准中，当以结构类型作为变量类型时，在变 量的声明语句中必须显式地出现struct关键字。因此，在我们这个“纯C环境中”编译这段代码，将会得到N多错误报告。  
因此，我们要把main()函数的第一行改为纯C形式的声明语句：  
struct A a;  
这样，再次编译，将是0错误0警告。  
总结  
最后，这个“纯C环境”只是辅助性的，要想学会编写“纯C程序”，首先要仔细学习标准C语言。不要指望这个环境能教会你写“纯C程序”，就像上面的例子，仅仅缺少一个关键字，在Visual Studio .NET 2003种将会得到这么多错误：  
e:\Workspace\C++\**PureC**\**PureC**.c(14): error C2065: “A” : 未声明的标识符  
e:\Workspace\C++\**PureC**\**PureC**.c(14): error C2146: 语法错误 : 缺少“;”(在标识符“a”的前面)  
e:\Workspace\C++\**PureC**\**PureC**.c(14): error C2144: 语法错误 : “;”的前面应有“;”  
e:\Workspace\C++\**PureC**\**PureC**.c(14): error C2144: 语法错误 : “;”的前面应有“;”  
e:\Workspace\C++\**PureC**\**PureC**.c(14): error C2143: 语法错误 : 缺少“;”(在“标识符”的前面)  
e:\Workspace\C++\**PureC**\**PureC**.c(14): error C2065: “a” : 未声明的标识符  
e:\Workspace\C++\**PureC**\**PureC**.c(15): error C2224: “.i”的左侧必须具有结构/联合类型  
如果你对标准C不是很了解的话，面对这么多错误肯定是无从下手的。因此，语言的学习是很重要的，环境只是一个辅助的功能。

# 虚函数的注意事项

1.构造函数  
构造函数不能是虚函数。创建派生类对象时，将调用派生类的构造函数，而不是基类的构造函数，然后，派生类的构造函数将使用基类的一个构造函数，这种顺序不同于继承机制。因此，派生类不继承基类的构造函数，所以将类构造函数声明为虚拟的没有什么意义．  
2.析构函数  
析构函数应当是虚函数，除非类不用做基类．例如，假设Employee是基类，Singer是派生类，并添加一个char\*成员，该成员指向由new分配的内存．当Singer对象过期时，必须调用～Singer()析构函数来释放内存．  
请看下面代码：  
Employee \*pe = new Singer; // legal because Employee is base for Singer  
.........  
delete pe; // ~Employee() or ~Singer()?  
如果使用默认的静态联编，delete语句将调用～Employee()析构函数．这将释放由Singer对象中的Employee部分指向的内存，但不会释放新的类成员指向的内存．但如果析构函数是虚拟的，则上述代码将先调用~Singer()析构函数释放由Singer组件指向的内存，然后，调用~Employee()析构函数来释放由Employee组件指向的内存．  
这意味着，即使基类不需要显示析构函数提供服务，也不应依赖于默认构造函数，而应提供虚拟析构函数，即使它不执行任何操作：  
virtual ~BaseClass(){}  
3. 友元  
友元不能是虚函数，因为友元不是类成员，而只有成员才能是虚函数．如果说由于这个原因引起了设计问题，可以通过让友元函数使用虚拟成员函数来解决．  
4.没有重新定义  
如果派生类没有重新定义函数，将使用该函数的基类版本．如果派生类位于派生链中，则将使用最新的虚函数版本，例外的情况是基类版本是隐藏的．  
5.重新定义隐藏方法  
假设创建了如下所示的代码：  
class Dwelling  
{  
public:  
virtual void showperks(int a) const;  
.....  
};  
class Hovel: public Dwelling  
{  
public:  
virtual void showperks() const;  
........  
};  
这将导致问题，可能会出现类似于下面这样的编译器警告：  
Warning: Hovel :: showperks(void) hides Dwelling::showperks(int)  
也可能不会出现警告．但结果怎么样，代码将具有如下含义：  
Hovel trump;  
trump.showperks(); // valid  
trump.showperks(5); //invalid  
新定义将showperks()定义一个不接受任何参数的函数．重新定义不会生成函数的两个重载版本，而是隐藏了接受一个int参数的基类版本．简而言之，重新定义继承的方法并不是重载．如果在派生类中重新定义函数，将不是使用相同的函数特征覆盖基类声明，而是隐藏同名的基类方法，不管参数特标如何．  
这引出两条经验规则：第一，如果重新定义继承的方法，应确保与原来的原型完全相同，但如果返回类型是基类引用或指针，则可以修改为指向派生类的引用或指针（这种例外是新出现的）．这种特征被称为返回类型协变(covariance of return type),因为允许返回类型随类类型的变化而变化：  
class Dwelling  
{  
public:  
// a base method   
virtual Dwelling &build(int n);  
......  
};  
class Hovel : public Dwelling  
{  
public:  
// a derived method with a covariant return type  
virtual Hovel &build(int n); // same function signature  
.........  
};  
注意，这种意外只适用返回值，而不适用于参数  
第二，如果基类声明被重载了，则应在派生类中重新定义所有的基类版本．  
class Dwelling  
{  
public:  
//three overloaded showperks()  
virtual void showperks (int a) const;  
virtual void showperks (double x) const;  
virtual void showperks() const;  
........  
};  
class Hovel: public Dwelling  
{  
public:  
// three redefined showperks()  
virtual void showperks(int a) const;  
virtual void showperks(double x) const;  
virtual void showperks() const;  
..........  
};  
如果只重新定义一个版本，则另外两个版本将被隐藏，派生类对象将无法使用它们．注意，如果不需要修改，则新定义可只调用基类版本．

# [C++中typedef struct与struct的区别](http://blog.csdn.net/quanchaoqq/article/details/7609220)

2012-05-28 14:4114人阅读[评论](http://blog.csdn.net/quanchaoqq/article/details/7609220#comments)(0)[收藏](javascript:void(0);)[举报](http://blog.csdn.net/quanchaoqq/article/details/7609220#report)

**第一篇：typedef struct与struct的区别**

1. 基本解释

typedef为C语言的关键字，作用是为一种数据类型定义一个新名字。这里的数据类型包括内部数据类型（int,char等）和自定义的数据类型（struct等）。

在编程中使用typedef目的一般有两个，一个是给变量一个易记且意义明确的新名字，另一个是简化一些比较复杂的类型声明。

至于typedef有什么微妙之处，请你接着看下面对几个问题的具体阐述。

2. typedef & 结构的问题

当用下面的代码定义一个结构时，编译器报了一个错误，为什么呢？莫非C语言不允许在结构中包含指向它自己的指针吗？请你先猜想一下，然后看下文说明：

typedef struct tagNode  
{  
　char \*pItem;  
　pNode pNext;  
} \*pNode;

答案与分析：

1、typedef的最简单使用

typedef long byte\_4;

给已知数据类型long起个新名字，叫byte\_4。

2、 typedef与结构结合使用

typedef struct tagMyStruct  
{  
　int iNum;  
　long lLength;  
} MyStruct;

这语句实际上完成两个操作：

1) 定义一个新的结构类型

struct tagMyStruct  
{  
　int iNum;  
　long lLength;  
};

分析：tagMyStruct称为“tag”，即“标签”，实际上是一个临时名字，struct 关键字和tagMyStruct一起，构成了这个结构类型，不论是否有typedef，这个结构都存在。

我们可以用struct tagMyStruct varName来定义变量，但要注意，**使用tagMyStruct varName来定义变量是不对的，因为struct 和tagMyStruct合在一起才能表示一个结构类型。**

2) typedef为这个新的结构起了一个名字，叫MyStruct。

typedef struct tagMyStruct MyStruct;

因此，MyStruct实际上相当于struct tagMyStruct，我们可以使用MyStruct varName来定义变量。

答案与分析

C语言当然允许在结构中包含指向它自己的指针，我们可以在建立链表等数据结构的实现上看到无数这样的例子，上述代码的根本问题在于typedef的应用。

根据我们上面的阐述可以知道：新结构建立的过程中遇到了pNext域的声明，类型是pNode，要知道pNode表示的是类型的新名字，那么在类型本身还没有建立完成的时候，这个类型的新名字也还不存在，也就是说这个时候编译器根本不认识pNode。

解决这个问题的方法有多种：

1)、

typedef struct tagNode  
{  
　char \*pItem;  
　struct tagNode \*pNext;  
} \*pNode;

2)、

typedef struct tagNode \*pNode;  
struct tagNode  
{  
　char \*pItem;  
　pNode pNext;  
};

注意：在这个例子中，你用typedef给一个还未完全声明的类型起新名字。C语言编译器支持这种做法。

3)、规范做法：

typedef uint32 (\* ADM\_READDATA\_PFUNC)( uint16\*, uint32 );

这个以前没有看到过,个人认为是预定义一个uint32的指针函数,uint16\*, uint32 为函数里的两个参数; 应该相当于#define uint32 (\* ADM\_READDATA\_PFUNC)( uint16\*, uint32 );

struct在代码中常见两种形式：  
struct A  
{  
//...  
};

struct  
{  
//...  
} A;  
这其实是两个完全不同的用法：  
前者叫做“结构体类型定义”，意思是：定义{}中的结构为一个名称是“A”的结构体。  
这种用法在typedef中一般是：  
typedef struct tagA //故意给一个不同的名字，作为结构体的实名  
{  
//...  
} A; //结构体的别名。

后者是结构体变量定义，意思是：以{}中的结构，定义一个名称为"A"的变量。这里的结构体称为匿名结构体，是无法被直接引用的。  
也可以通过typedef为匿名结构体创建一个别名，从而使得它可以被引用：  
typedef struct  
{  
//...  
} A; //定义匿名结构体的别名为A

**第二篇：在C和C++中struct和typedef struct的区别**

在C和C++有三种定义结构的方法。

typedef struct {

int data;

int text;

} S1;

//这种方法可以在c或者c++中定义一个S1结构

struct S2 {

int data;

int text;

};

// 这种定义方式只能在C++中使用，而如果用在C中，那么编译器会报错

struct {

int data;

int text;

} S3;

这种方法并没有定义一个结构，而是定义了一个s3的结构变量，编译器会为s3内存。

void main()

{

S1 mine1;// OK ,S1 是一个类型

S2 mine2;// OK,S2 是一个类型

S3 mine3;// OK,S3 不是一个类型

S1.data = 5;// ERRORS1 是一个类型

S2.data = 5;// ERRORS2 是一个类型

S3.data = 5;// OKS3是一个变量

}

另外，对与在结构中定义结构本身的变量也有几种写法

struct S6 {

S6\* ptr;

};

// 这种写法只能在C++中使用

typedef struct {

S7\* ptr;

} S7;

// 这是一种在C和C++中都是错误的定义

如果在C中，我们可以使用这样一个“曲线救国的方法“

typedef struct tagS8{

tagS8 \* ptr;

} S8;

**第三篇：struct和typedef struct**

分三块来讲述：  
1 首先：  
在C中定义一个结构体类型要用typedef:  
typedef struct Student  
{  
int a;  
}Stu;  
于是在声明变量的时候就可：Stu stu1;  
**如果没有typedef就必须用struct Student stu1;来声明  
这里的Stu实际上就是struct Student的别名。**  
另外这里也可以不写Student（于是也不能struct Student stu1;了）  
typedef struct  
{  
int a;  
}Stu;  
**但在c++里很简单，直接  
struct Student  
{  
int a;  
};  
于是就定义了结构体类型Student，声明变量时直接Student stu2；**  
===========================================  
2其次：  
在c++中如果用typedef的话，又会造成区别：  
struct Student  
{  
int a;  
}stu1;//stu1是一个变量  
typedef struct Student2  
{  
int a;  
}stu2;//stu2是一个结构体类型  
使用时可以直接访问stu1.a  
但是stu2则必须先 stu2 s2;  
然后 s2.a=10;  
===========================================  
3 掌握上面两条就可以了，不过最后我们探讨个没多大关系的问题  
如果在c程序中我们写：  
typedef struct  
{  
int num;  
int age;  
}aaa,bbb,ccc;  
这算什么呢？  
我个人观察编译器（VC6）的理解，这相当于  
typedef struct  
{  
int num;  
int age;  
}aaa；  
typedef aaa bbb;  
typedef aaa ccc;  
也就是说aaa,bbb,ccc三者都是结构体类型。声明变量时用任何一个都可以,在c++中也是如此。但是你要注意的是这个在c++中如果写掉了typedef关键字，那么aaa，bbb，ccc将是截然不同的三个对象。

**第四篇：C/C++中typedef struct和struct的用法**

struct \_x1 { ...}x1; 和 typedef struct \_x2{ ...} x2; 有什么不同？  
  
其实, 前者是定义了类\_x1和\_x1的对象实例x1, 后者是定义了类\_x2和\_x2的类别名x2,  
  
所以它们在使用过程中是有取别的.请看实例1.  
  
[知识点]  
  
**结构体也是一种数据类型**, 可以使用结构变量, 因此, 象其它 类型的变量一样, 在使用结构变量时要先对其定义。  
  
定义结构变量的一般格式为:  
  
struct 结构名  
  
{  
  
类型 变量名;  
  
类型 变量名;  
  
...  
  
} 结构变量;  
  
结构名是结构的标识符不是变量名。  
  
另一种常用格式为:  
  
typedef struct 结构名  
  
{  
  
类型 变量名;  
  
类型 变量名;  
  
...  
  
} 结构别名;  
  
另外注意: **在C中，struct不能包含函数。在C++中，对struct进行了扩展，可以包含函数。**  
  
======================================================================  
  
实例1: struct.cpp  
  
#include <iostream>  
  
using namespace std;  
  
typedef struct \_point{  
  
int x;  
  
int y;  
  
}point; //定义类，给类一个别名  
  
struct \_hello{  
  
int x,y;  
  
} hello; //同时定义类和对象  
  
int main()  
  
{  
  
point pt1;  
  
pt1.x = 2;  
  
pt1.y = 5;  
  
cout<< "ptpt1.x=" << pt1.x << "pt.y=" <<pt1.y <<endl;  
  
//hello pt2;  
  
//pt2.x = 8;  
  
//pt2.y =10;  
  
//cout<<"pt2pt2.x="<< pt2.x <<"pt2.y="<<pt2.y <<endl;  
  
//上面的hello pt2;这一行编译将不能通过. 为什么?  
  
//因为hello是被定义了的对象实例了.  
  
//正确做法如下: 用hello.x和hello.y  
  
hello.x = 8;  
  
hello.y = 10;  
  
cout<< "hellohello.x=" << hello.x << "hello.y=" <<hello.y <<endl;  
  
return 0;  
  
}

**第五篇：问答**

**Q：**用struct和typedef struct 定义一个结构体有什么区别？为什么会有两种方式呢？

struct Student  
{  
int a;  
} stu;  
typedef struct Student2  
{  
int a;  
}stu2;

**A：**

事实上，这个东西是从C语言中遗留过来的，typedef可以定义新的复合类型或给现有类型起一个别名，在C语言中，如果你使用  
struct xxx  
{  
}; 的方法，使用时就必须用 struct xxx var 来声明变量，而使用  
typedef struct  
{  
}的方法 就可以写为 xxx var;  
不过在C++中已经没有这回事了，无论你用哪一种写法都可以使用第二种方式声明变量，这个应该算是C语言的糟粕。

# 回调函数

回调[函数](http://baike.baidu.com/view/15061.htm)就是一个通过[函数指针](http://baike.baidu.com/view/1604730.htm)调用的函数。如果你把函数的[指针](http://baike.baidu.com/view/159417.htm)（地址）作为[参数传递](http://baike.baidu.com/view/2691131.htm)给另一个函数，当这个指针被用为调用它所指向的函数时，我们就说这是回调函数。回调函数不是由该函数的实现方直接调用，而是在特定的事件或条件发生时由另外的一方调用的，用于对该事件或条件进行响应。

## 回调函数实现的机制是

　　⑴定义一个回调[函数](http://baike.baidu.com/view/15061.htm)；

　　⑵提供函数实现的一方在初始化的时候，将回调函数的函数[指针](http://baike.baidu.com/view/159417.htm)注册给调用者；

　　⑶当特定的事件或条件发生的时候，调用者使用[函数指针](http://baike.baidu.com/view/1604730.htm)调用回调函数对事件进行处理。

## [编辑本段](http://baike.baidu.com/view/414773.htm)为什么要使用回调函数

　　因为可以把调用者与被调用者分开，所以调用者不关心谁是被调用者。它只需知道存在一个具有特定原型和限制条件的[被调用函数](http://baike.baidu.com/view/5295610.htm)。

　　想知道回调[函数](http://baike.baidu.com/view/15061.htm)在实际中有什么作用？先假设有这样一种情况：我们要编写一个库，它提供了某些[排序算法](http://baike.baidu.com/view/297739.htm)的实现（如[冒泡排序](http://baike.baidu.com/view/254413.htm)、[快速排序](http://baike.baidu.com/view/115472.htm)、[shell排序](http://baike.baidu.com/view/549624.htm)、shake排序等等），为了能让库更加通用，不想在函数中嵌入排序逻辑，而让使用者来实现相应的逻辑；或者，能让库可用于多种[数据类型](http://baike.baidu.com/view/675645.htm)（int、float、string），此时，该怎么办呢？可以使用[函数指针](http://baike.baidu.com/view/1604730.htm)，并进行回调。

　　回调可用于通知机制。例如，有时要在A程序中设置一个[计时器](http://baike.baidu.com/view/928508.htm)，每到一定时间，A程序会得到相应的通知，但通知机制的实现者对A程序一无所知。那么，就需一个具有特定原型的函数指针进行回调，通知A程序事件已经发生。实际上，API使用一个回调函数SetTimer()来通知计时器。如果没有提供回调函数，它还会把一个消息发往程序的[消息队列](http://baike.baidu.com/view/262473.htm)。

　　另一个使用回调机制的API函数是EnumWindow（），它枚举屏幕上所有的顶层窗口，每个窗口都可以通过它调用另一个程序提供的函数，并传递窗口的处理程序。例如：如果被调用者返回一个值，就继续进行迭代；否则，退出。EnumWindow()并不关心被调用者在何处，也不关心被调用者用它传递的处理程序做了什么，它只关心返回值，因为基于返回值，它将继续执行或退出。

　　不管怎么说，回调函数是继承自C语言的。在C++中，应只在与C代码建立接口或与已有的回调接口打交道时，才使用回调函数。除了上述情况，在C++中应使用虚拟方法或函数符（functor），而不是回调函数。

## [编辑本段](http://baike.baidu.com/view/414773.htm)简单的回调函数实现

### 代码实现

　　下面创建了一个sort.dll的[动态链接库](http://baike.baidu.com/view/887.htm)，它导出了一个名为CompareFunction的类型--typedef int (\_\_stdcall \*CompareFunction)(const byte\*,const byte\*），它就是回调[函数](http://baike.baidu.com/view/15061.htm)的类型。另外，它也导出了两个方法：Bubblesort（）和Quicksort（），这两个方法原型相同，但实现了不同的[排序算法](http://baike.baidu.com/view/297739.htm)。

　　void DLLDIR \_\_stdcall Bubblesort(byte\* array,int size,int elem\_size,CompareFunction cmpFunc);

　　void DLLDIR \_\_stdcall Quicksort(byte\* array,int size,int elem\_size,CompareFunction cmpFunc);

　　这两个函数接受以下参数：

　　·byte \* array：指向元素[数组](http://baike.baidu.com/view/209670.htm)的[指针](http://baike.baidu.com/view/159417.htm)（任意类型）。

　　·int size：数组中元素的个数。

　　·int elem\_size：数组中一个元素的大小，以[字节](http://baike.baidu.com/view/60408.htm)为单位。

　　·CompareFunction cmpFunc：带有上述原型的指向回调函数的指针。

　　这两个函数都会对数组进行某种排序，但每次都需决定两个元素哪个排在前面，而函数中有一个回调函数，其地址是作为一个[参数传递](http://baike.baidu.com/view/2691131.htm)进来的。对编写者来说，不必介意函数在何处实现，或它怎样被实现的，所需在意的只是两个用于比较的元素的地址，并返回以下的某个值（库的编写者和使用者都必须遵守这个约定）：

　　·-1：如果第一个元素较小，那它在已排序好的数组中，应该排在第二个元素前面。

　　·0：如果两个元素相等，那么它们的相对位置并不重要，在已排序好的数组中，谁在前面都无所谓。

　　·1：如果第一个元素较大，那在已排序好的数组中，它应该排第二个元素后面。

　　基于以上约定，函数Bubblesort（）的实现如下，Quicksort（）就稍微复杂一点：

　　void DLLDIR \_\_stdcall Bubblesort(byte\* array,int size,int elem\_size,CompareFunction cmpFunc)

　　{

　　for(int i=0; i < size; i++)

　　{

　　for(int j=0; j < size-i-1; j++)

　　{

　　//回调比较函数

　　if(1 == (\*cmpFunc)(array+j\*elem\_size,array+(j+1)\*elem\_size))

　　{

　　//两个相比较的元素相交换

　　byte\* temp = new byte[elem\_size];

　　memcpy(temp,array+j\*elem\_size,elem\_size);

　　memcpy(array+j\*elem\_size,array+(j+1)\*elem\_size,elem\_size);

　　memcpy(array+(j+1)\*elem\_size,temp,elem\_size);

　　delete [] temp;

　　}

　　}

　　}

　　}

　　注意：因为实现中使用了memcpy（），所以函数在使用的[数据类型](http://baike.baidu.com/view/675645.htm)方面，会有所局限。

　　对使用者来说，必须有一个回调函数，其地址要传递给Bubblesort（）函数。下面有二个简单的示例，一个比较两个整数，而另一个比较两个字符串：

　　int \_\_stdcall CompareInts(const byte\* velem1,const byte\* velem2)

　　{

　　int elem1 = \*(int\*)velem1;

　　int elem2 = \*(int\*)velem2;

　　if(elem1 < elem2)

　　return -1;

　　if(elem1 > elem2)

　　return 1;

　　return 0;

　　}

　　int \_\_stdcall CompareStrings(const byte\* velem1,const byte\* velem2)

　　{

　　const char\* elem1 = (char\*)velem1;

　　const char\* elem2 = (char\*)velem2;

　　return strcmp(elem1,elem2);

　　}

　　下面另有一个程序，用于测试以上所有的代码，它传递了一个有5个元素的数组给Bubblesort（）和Quicksort（），同时还传递了一个指向回调函数的指针。（使用byte类型需包含头文件windows.h，或typedefunsigned char byte)

　　int main(int argc,char\* argv[])

　　{

　　int i;

　　int array[] = {5432,4321,3210,2109,1098};

　　cout << "Before sorting ints with Bubblesort\n";

　　for(i=0; i < 5; i++)

　　cout << array[i]<< '\n';

　　Bubblesort((byte\*)array,5,sizeof(array[0]),&CompareInts);

　　cout << "After the sorting\n";

　　for(i=0; i < 5; i++)

　　cout << array[i]<< '\n';

　　const char str[5][10] = {"estella","danielle","crissy","bo","angie"};

　　cout << "Before sorting strings with Quicksort\n";

　　for(i=0; i < 5; i++)

　　cout << str[i]<< '\n';

　　Quicksort((byte\*)str,5,10,&CompareStrings);

　　cout << "After the sorting\n";

　　for(i=0; i < 5; i++)

　　cout << str[i]<< '\n';

　　return 0;

　　}

　　如果想进行降序排序（大元素在先），就只需修改回调函数的代码，或使用另一个回调函数，这样编程起来灵活性就比较大了。

### 调用约定

　　上面的代码中，可在[函数](http://baike.baidu.com/view/15061.htm)原型中找到\_\_stdcall，因为它以双下划线打头，所以它是一个特定于[编译器](http://baike.baidu.com/view/487018.htm)的扩展，说到底也就是[微软](http://baike.baidu.com/view/2353.htm)的实现。任何支持开发基于Win32的程序都必须支持这个扩展或其等价物。以\_\_stdcall标识的函数使用了标准调用约定，为什么叫标准约定呢，因为所有的Win32 API（除了个别接受可变参数的除外）都使用它。标准调用约定的函数在它们返回到调用者之前，都会从[堆栈](http://baike.baidu.com/view/93201.htm)中移除掉参数，这也是Pascal的标准约定。但在C/C++中，调用约定是调用者负责清理堆栈，而不是[被调用函数](http://baike.baidu.com/view/5295610.htm)；为强制函数使用C/C++调用约定，可使用\_\_cdecl。另外，可变参数函数也使用C/C++调用约定。

　　Windows[操作系统](http://baike.baidu.com/view/880.htm)采用了标准调用约定（Pascal约定），因为其可减小代码的体积。这点对早期的Windows来说非常重要，因为那时它运行在只有640KB内存的电脑上。

　　如果你不喜欢\_\_stdcall，还可以使用CALLBACK宏，它定义在windef.h中：

　　#define CALLBACK \_\_stdcallor

　　#define CALLBACK PASCAL //而PASCAL在此被#defined成\_\_stdcall

　　作为回调函数的C++方法

　　因为平时很可能会使用到C++编写代码，也许会想到把回调函数写成类中的一个方法，但先来看看以下的代码：

　　class CCallbackTester

　　{

　　public:

　　int CALLBACK CompareInts(const byte\* velem1,const byte\* velem2);

　　};

　　Bubblesort((byte\*)array,5,sizeof(array[0]),

　　&CCallbackTester::CompareInts);

　　如果使用微软的编译器，将会得到下面这个编译错误：

　　error C2664: ’Bubblesort’ : cannot convert parameter 4 from ’int (\_\_stdcall CCallbackTester::\*)(const unsigned char \*,const unsigned char \*）’ to ’int (\_\_stdcall \*)(const unsigned char \*,const unsigned char \*）’ There is no context in which this conversion is possible

这是因为非[静态成员](http://baike.baidu.com/view/534160.htm)函数有一个额外的参数：this[指针](http://baike.baidu.com/view/159417.htm)，这将迫使你在成员函数前面加上static。

# 运算符优先级

目录

[语言操作符](http://baike.baidu.com/view/262524.htm" \l "1)

[三个优先级](http://baike.baidu.com/view/262524.htm#2)

[算术运算符](http://baike.baidu.com/view/262524.htm#3)

[C++运算符优先级](http://baike.baidu.com/view/262524.htm#4)

[C#中的优先级](http://baike.baidu.com/view/262524.htm#5)

## [编辑本段](http://baike.baidu.com/view/262524.htm)语言操作符

　　c语言的操作符共有15个优先级，如下：

　　Operators Associativity

　　------------------------------------------------------------------

　　运算符 结合性

　　------------------------------------------------------------------

　　() [] -> . left to right

　　! ~ ++ -- + - \* sizeof(type) **right to left**

　　\* / % left to right

　　+ - left to right

　　<< >> left to right

　　< <= > >= left to right

　　== != left to right

　　& left to right

　　^ left to right

　　| left to right

　　&& left to right

　　|| left to right

　　?: **right to left**

= += -= \*= /= %= &= ^=

|= <<= >>= **right to left**

　　, left to right

## [编辑本段](http://baike.baidu.com/view/262524.htm)三个优先级

　　优先级从上到下依次递减，最上面具有最高的优先级，逗号操作符具有最低的优先级。

　　所有的优先级中，只有三个优先级是从右至左结合的，它们是[单目运算符](http://baike.baidu.com/view/4072122.htm)、条件运算符、[赋值运算符](http://baike.baidu.com/view/1589776.htm)。其它的都是从左至右结合。

　　具有最高优先级的其实并不算是真正的运算符，它们算是一类特殊的操作。()是与函数相关，[]与[数组](http://baike.baidu.com/view/209670.htm)相关，而－>及.是取结构成员。

　　其次是单目运算符，所有的单目运算符具有相同的优先级，因此在我认为的 真正的运算符中它们具有最高的优先级，又由于它们都是从右至左结合的，因此\*p++与\*(p++)等效是毫无疑问的。

## [编辑本段](http://baike.baidu.com/view/262524.htm)算术运算符

　　接下来是算术运算符，\*、/、%的优先级当然比+、－高了。

[移位运算符](http://baike.baidu.com/view/3103728.htm)紧随其后。

　　其次的关系运算符中，< <= > >=要比 == !=高一个级别，不大好理解。

　　所有的逻辑操作符都具有不同的优先级（[单目运算符](http://baike.baidu.com/view/4072122.htm)除外，！和~）

　　逻辑位操作符的"与"比"或"高，而"异或"则在它们之间。

　　跟在其后的&&比||高。

　　接下来的是条件运算符，[赋值运算符](http://baike.baidu.com/view/1589776.htm)及逗号运算符。

　　在C语言中，只有4个运算符规定了运算方向，它们是&&、| |、条件运算符及赋值运算符。

　　&&、| |都是先计算左边[表达式](http://baike.baidu.com/view/420676.htm)的值，当左边表达式的值能确定整个表达式的值时，就不再计算右边表达式的值。如 a = 0 && b; &&运算符的左边位为0，则右边表达式b就不再判断。

　　在条件运算符中。如a?b:c；先判断a的值，再根据a的值对b或c之中的一个进行求值。

　　赋值表达式则规定先对右边的表达式求值，因此使 a = b = c = 6;成为可能。

## [编辑本段](http://baike.baidu.com/view/262524.htm)C++运算符优先级

|  |  |  |  |
| --- | --- | --- | --- |
| Operator | Description | Example |  |
| **Group 1** (N) | 没有结合型 |  |  |
| :: | 域操作符 | Class::age = 2; | NO |
| **Group 2** |  |  |  |
| () | 函数调用 | isdigit('1') | YES |
| () | 成员初始化 | c\_tor(int x, int y) : \_x(x) | YES |
| [] | 数组入口 | array[4] = 2; | YES |
| -> | 指针访问成员 | ptr->age = 34; | YES |
| . | 对象访问成员 | obj.age = 34; | NO |
| ++ | 后缀递增 | for( int i = 0; i < 10; i++ ) | YES |
| -- | 会追递减 | for( int i = 10; i > 0; i-- ) | YES |
| const\_cast | Special cast | const\_cast<type\_to>(type\_from); | NO |
| dynamic\_cast | Special cast | dynamic\_cast<type\_to>(type\_from); | NO |
| static\_cast | Special cast | static\_cast<type\_to>(type\_from); | NO |
| reinterpret\_cast | Special cast | reinterpret\_cast<type\_to>(type\_from); | NO |
| typeid | 运行时类型信息 | cout &laquo; typeid(var).name();  cout &laquo; typeid(type).name(); | NO |
| **Group 3** (RL) |  |  |  |
| ! | 逻辑反 | if( !done ) … | YES |
| ~ | 按位取反 | flags = ~flags; | YES |
| ++ | 前缀递增 | for( i = 0; i < 10; ++i ) cout << i; | YES |
| -- | 前缀递减 | for( i = 10; i > 0; --i ) cout << i; | YES |
| - | 一元负 | int i = -1; | YES |
| + | 一元加 | int i = +1; | YES |
| \* | 解指针 | int data = \*intPtr; | YES |
| & | 取地址 | int \*intPtr = &data; | YES |
| new | 动态内存申请 |  | YES |
| new [] | 动态数组内存申请 | long \*array = new long[n]; | YES |
| delete | Deallocating the memory | delete pVar; | YES |
| delete [] | Deallocating the memory of array | delete [] array; | YES |
| (type) | Cast to a given type | int i = (int) floatNum; | YES |
| sizeof | 返回对象大小 |  | NO |
| **Group 4** |  |  |  |
| ->\* | Member pointer selector | ptr->\*var = 24; | YES |
| .\* | Member object selector | obj.\*var = 24; | NO |
| **Group 5** |  |  |  |
| \* | Multiplication | int i = 2 \* 4; | YES |
| / | Division | float f = 10.0 / 3.0; | YES |
| % | Modulus | int rem = 4 % 3; | YES |
| **Group 6** |  |  |  |
| + | Addition | int i = 2 + 3; | YES |
| - | Subtraction | int i = 5 - 1; | YES |
| **Group 7** |  |  |  |
| << | Bitwise shift left | int flags = 33 << 1; | YES |
| >> | Bitwise shift right | int flags = 33 >> 1; | YES |
| **Group 8** |  |  |  |
| < | Comparison less-than | if( i < 42 ) … | YES |
| <= | Comparison less-than-or-equal-to | if( i <= 42 ) ... | YES |
| > | Comparison greater-than | if( i > 42 ) … | YES |
| >= | Comparison greater-than-or-equal-to | if( i >= 42 ) ... | YES |
| **Group 9** |  |  |  |
| == | Comparison equal-to | if( i == 42 ) ... | YES |
| != | Comparison not-equal-to | if( i != 42 ) … | YES |
| **Group 10** |  |  |  |
| & | Bitwise AND | flags = flags & 42; | YES |
| **Group 11** |  |  |  |
| ^ | Bitwise exclusive OR (XOR) | flags = flags ^ 42; | YES |
| **Group 12** |  |  |  |
| | | Bitwise inclusive (normal) OR | flags = flags | 42; | YES |
| **Group 13** |  |  |  |
| && | Logical AND | if( conditionA && conditionB ) … | YES |
| **Group 14** |  |  |  |
| || | Logical OR | if( conditionA || conditionB ) ... | YES |
| **Group 15** (RL) |  |  |  |
| ? : | Ternary conditional (if-then-else) | int i = (a > b) ? a : b; | NO |
| **Group 16** (RL) |  |  |  |
| = | Assignment operator | int a = b; | YES |
| += | Increment and assign | a += 3; | YES |
| -= | Decrement and assign | b -= 4; | YES |
| \*= | Multiply and assign | a \*= 5; | YES |
| /= | Divide and assign | a /= 2; | YES |
| %= | Modulo and assign | a %= 3; | YES |
| &= | Bitwise AND and assign | flags &= new\_flags; | YES |
| ^= | Bitwise exclusive or (XOR) and assign | flags ^= new\_flags; | YES |
| |= | Bitwise normal OR and assign | flags |= new\_flags; | YES |
| <<= | Bitwise shift left and assign | flags <<= 2; | YES |
| >>= | Bitwise shift right and assign | flags >>= 2; | YES |
| **Group 17** |  |  |  |
| throw | throw exception | throw EClass(“Message”); | NO |
| **Group 18** |  |  |  |
| , | Sequential evaluation operator | for( i = 0, j = 0; i < 10; i++, j++ ) | YES |

# [C++中的vector使用范例](http://blog.csdn.net/tjh666/article/details/1604119)

分类： [C/C++](http://blog.csdn.net/tjh666/article/category/302082)2007-05-11 10:1827307人阅读[评论](http://blog.csdn.net/tjh666/article/details/1604119#comments)(8)[收藏](javascript:void(0);)[举报](http://blog.csdn.net/tjh666/article/details/1604119#report)

前几天一个同学问我c++里面的vector的使用，我研究了一下，先总结如下：

1.vector 的数据的存入和输出：

#include<stdio.h>  
#include<vector>  
#include <iostream>

using namespace std;

void main()

{

int i = 0;

vector<int> v;

for( i = 0; i < 10; i++ )

{

v.push\_back( i );//把元素一个一个存入到vector中

}

for( i = 0; i < v.size(); i++ )//v.size() 表示vector存入元素的个数

{

cout << v[ i ] << " "; //把每个元素显示出来

}

cont << endl;

}

注：你也可以用v.begin()和v.end() 来得到vector开始的和结束的元素地址的指针位置。你也可以这样做：

vector<int>::iterator iter;

for( iter = v.begin(); iter != v.end(); iter++ )

{  
cout << \*iter << endl;  
}

2. 对于二维vector的定义。

1)定义一个10个vector元素，并对每个vector符值1-10。

#include<stdio.h>  
#include<vector>  
#include <iostream>

using namespace std;

void main()  
{  
int i = 0, j = 0;

//定义一个二维的动态数组，有10行，每一行是一个用一个vector存储这一行的数据。

所以每一行的长度是可以变化的。之所以用到vector<int>(0)是对vector初始化，否则不能对vector存入元素。  
vector< vector<int> > Array( 10, vector<int>(0) );

for( j = 0; j < 10; j++ )  
{  
for ( i = 0; i < 9; i++ )  
{  
Array[ j ].push\_back( i );  
}  
}

for( j = 0; j < 10; j++ )  
{  
for( i = 0; i < Array[ j ].size(); i++ )  
{  
cout << Array[ j ][ i ] << " ";  
}  
cout<< endl;  
}  
}

2)定义一个行列都是变化的数组。

#include<stdio.h>  
#include<vector>  
#include <iostream>

using namespace std;

void main()  
{  
int i = 0, j = 0;

vector< vector<int> > Array;  
vector< int > line;  
for( j = 0; j < 10; j++ )  
{  
Array.push\_back( line );//要对每一个vector初始化，否则不能存入元素。  
for ( i = 0; i < 9; i++ )  
{  
Array[ j ].push\_back( i );  
}  
}

for( j = 0; j < 10; j++ )  
{  
for( i = 0; i < Array[ j ].size(); i++ )  
{  
cout << Array[ j ][ i ] << " ";  
}  
cout<< endl;  
}  
}

上面就是我对vector使用的总结，更深入的使用，大家查查vector的手册吧。欢迎批评指正。

# [C++中cin.get和cin.getline()的区别](http://registerboy.pixnet.net/blog/post/12761086)

分享: [7Headlines](javascript:jQuery.getScript('http://www.7headlines.com/static/js/clip.min.js?random=691.3958971600074');)facebookPLURKtwitter

cin<<

cin.get()

cin.getline()

三个都是C++中的输入函数，不同点在于

cin<<的结束符有enter space tab

cin.get()的结束符只有enter，但会把enter放入队列等待

cin.getline()的结束符也只有enter，但不会把enter放入队列

cin.get() 有三种格式

无参，一参数，二参数

cin.get() , cin.get(char ch) , cin.get(array\_name,Arsize)

cin.getline()只有一种格式

cin.getline(array\_name,Arsize)

Example:

main()  
{  
const int arsize=20;  
char name1[arsize];  
char name2[arsize];  
int age;  
cout<<"what is your first name?";

cin.getline(name1,arsize);  
//cin.get(name1,arsize).get(); cin.get()可用来装载空enter  
  
cout<<"what is your last name?";  
cin.get(name2,arsize);  
  
cout<<"what is your age?";  
cin>>age;  
  
cout<<"name:"<<name1<<','<<name2<<'\n'  
<<"age:"<<age<<'\n';

}

cin.get遇到结束标志字符后,最后指针指向结束标志字符.

cin.getline指向结束标志符后一个字符;

比如:

#include<iostream>  
using namespace std;

int main()  
{  
char ch[20];  
cin>>ch;  
cout<<ch<<endl;  
cin.getline(ch,20,'/');  
cout<<ch<<endl;  
cin.getline(ch,20,'/');  
cout<<ch<<endl;  
return 0;  
}

输入:i like c++./i study c++./i am happy.回车

输出结果为:

i

\_like c++. //\_表示空格

i study c++.

因此<C++程序设计>这本书中,就错了.

如果把第一个cin.getline改为cin.get,

结果为:

i

\_like c++. //\_表示空格

//这行为空

可以看到,第三行为空,因为使用cin.get以后,字符指针指向结束标志字符,当用cin.getline读取时,一开始就遇到结果标志符,因为第三行为空.  
  
  
cin.get()也是输入一行（同cin.getline()），但是区别就是，不输出分隔符~ 跟以上兩種說法的意思是一致的

# c++强制转化

强制转化四种类型可能很多人都常常忽略就象我一样，但是有时还是比较有用的。不了解的建议看看，一些机制我也不是十分了解，只是将一些用法写出来让大家看看。

                                                            2004-11-27 9:00

强制转化无论从语法还是语意上看，都是c++中最难看的特征之一。但是基于c风格的转化的语义的不明确性及其一些潜在问题。强制类型转化最终还是被c++接受了。

1.static\_cast运算符号

static\_cast<T>(e),stroustrup让我们可以把它看成隐含转换的显示的逆运算。这个是有一定道理的，基于隐式转化的对象类型我们可以使用static\_cast转化运算符号。它是静态的检测，无法运行时检测类型，在继承中尤为突出。

使用范围

<1>用于所有系统类型之间转化，不能用于系统类型指针类型转化

  double t\_d = 0;

int t\_i= static\_cast<int>(t\_d); //是合法的转化

而企图将double\*->int\*是不允许的

<2>用于继承类之间的转化（含指针），不能用于其他没有隐式转化的对象类型之间的转化

继承举例:

class x

{

};

class y: public x

{

};

使用:x t\_o\_x;

y t\_o\_y = static\_cast<y>(t\_o\_x); //x\* y\*转化也可以进行因为x,y继承关

//系，类型可以自动隐式转化使用

   隐式转化举例:

class x

{

};

class y

{

public:

    y( x i\_x ) {}

};

    x t\_o\_x;

     y t\_o\_y = static\_cast<y>(t\_o\_x); //大家看到y构造函数可以对于x类型隐式转化

//所以可以将x->y，如果企图将y->x会报错

2.reinterpret\_cast 运算

主要用于对于类型指针类型的强制转化，some\_type\* -> special\_type\*这样转化，类型信息可以是不完全的。它允许将任意指针转化到其他类型指针，也允许任意整数类型到任意指针类型转化(BT)。这样导致的结果是极其不安全的，不能安全的应用于其他目的，除非转化到原来类型。

<1> 使用所有整形可以转化为任意类型的指针(指针是4字节的long的东东，那么机器就认为同类型就是可以转化)

int c;

x\* p = reinterpret\_cast<x\*>(c); //x是自定义的任意类型，当然包括系统类型

<2> 可以对于任意类型指针之间转化

y\* c;

x\* p = reinterpret\_cast<x\*>(c);//x,y代表所有自定义或系统类型

大家可以看到reinterpret\_cast的转化是极度的不负责任的，他只管转化不检测是否可以转化。

<3> const\_cast运算符号

这个很简单从名字大家可以看出来，仅仅为了去掉或着加上const修饰符号。但是对于本身定义时为const的类型，即使你去掉const性，在你操作这片内容时候也要小心，只能r不能w操作，否则还是会出错。

const char\* p = "123";

char\* c = const\_cast<char\*>(p);

c[0] = 1;  //表面上通过编译去掉了const性，但是操作其地址时系统依然不允许这

//么做。这是一个漏洞吧

<4> dynamic\_cast运算符号

Scott Mayers将其描述为用来执行继承体系中：安全的向下转型或者跨系转型动作。也就是说你可以，用dynamic\_cast将 指向base class的指针或引用转型为 指向子类的对象的指针或引用。

class B {};  //polymorphic类型含virtual才能dynamic\_cast

class D: public B {}

void f( B\* pb )

{

    D\* pd1 = dynamic\_cast<D\*>(pb);//如果pb为d类型正确返回，如果不是返回0

    D\* pd2 = static\_cast<D\*>(pb); //不管怎么样都返回指针有可能指向不合适的对

//象，因为static仅仅静态检测，不能得到运

//行时对象的信息是否真正为D类型

}

反正大家在使用知道怎么用就ok了，c++强制转化在模板中还是非常有用的，其他时候本人也喜欢用c的转化方便。^\_^  

# TCP/IP协议

#### 百科名片

TCP/IP协议层级模型

Transmission Control Protocol/Internet Protocol的简写，中译名为传输控制协议/[因特网](http://baike.baidu.com/view/1706.htm)互联协议，又名网络[通讯协议](http://baike.baidu.com/view/278358.htm)，是Internet最基本的协议、Internet国际[互联网](http://baike.baidu.com/view/6825.htm)络的基础，由[网络层](http://baike.baidu.com/view/239600.htm)的IP协议和[传输层](http://baike.baidu.com/view/239605.htm)的TCP协议组成。TCP/IP 定义了电子设备如何连入因特网，以及数据如何在它们之间传输的标准。协议采用了4层的层级结构，每一层都呼叫它的下一层所提供的网络来完成自己的需求。通俗而言：TCP负责发现传输的问题，一有问题就发出信号，要求重新传输，直到所有[数据安全](http://baike.baidu.com/view/2308446.htm)正确地传输到目的地。而IP是给因特网的每一台电脑规定一个地址。

目录

## 层次概述

　　从协议[分层模型](http://baike.baidu.com/view/547346.htm)方面来讲，TCP/IP由四个层次组成：[网络接口层](http://baike.baidu.com/view/4022168.htm)、[网络层](http://baike.baidu.com/view/239600.htm)、[传输层](http://baike.baidu.com/view/239605.htm)、[应用层](http://baike.baidu.com/view/239619.htm)。

　　TCP/IP[协议](http://baike.baidu.com/view/36190.htm)并不完全符合[OSI](http://baike.baidu.com/view/113948.htm)的七层参考模型。OSI（Open System Interconnect）是传统的[开放式系统](http://baike.baidu.com/view/1171987.htm)互连参考模型，是一种[通信协议](http://baike.baidu.com/view/185322.htm)的7层抽象的参考模型，其中每一层执行某一特定任务。该模型的目的是使各种硬件在相同的层次上相互通信。这7层是：[物理层](http://baike.baidu.com/view/239585.htm)、[数据链路层](http://baike.baidu.com/view/239592.htm)、网络层、传输层、[会话层](http://baike.baidu.com/view/239612.htm)、[表示层](http://baike.baidu.com/view/239615.htm)和应用层。而TCP/IP[通讯协议](http://baike.baidu.com/view/278358.htm)采用了4层的层级结构，每一层都呼叫它的下一层所提供的网络来完成自己的需求。由于ARPNET的设计者注重的是[网络互联](http://baike.baidu.com/view/1359266.htm)，允许[通信子网](http://baike.baidu.com/view/676314.htm)（网络接口层）采用已有的或是将来有的各种协议，所以这个层次中没有提供专门的协议。实际上，TCP/IP协议可以通过网络接口层连接到任何网络上，例如X.25交换网或[IEEE](http://baike.baidu.com/view/3933.htm)802局域网。

|  |  |
| --- | --- |
| TCP/IP结构对应OSI结构 | |
| **TCP/IP** | **OSI** |
| 应用层 | 应用层  表示层  会话层 |
| 主机到主机层（TCP）（又称传输层） | 传输层 |
| 网络层（IP）(又称互联层) | 网络层 |
| 网络接口层（又称链路层） | 数据链路层 |
| 物理层 |

### 网络接口层

　　物理层是定义物理介质的各种特性：

　　1、[机械特性](http://baike.baidu.com/view/1590129.htm)。

　　2、电子特性。

　　3、[功能特性](http://baike.baidu.com/view/3564002.htm)。

　　4、规程特性。

　　数据链路层是负责接收IP数据包并通过网络发送之，或者从网络上接收物理[帧](http://baike.baidu.com/view/24183.htm)，抽出IP[数据包](http://baike.baidu.com/view/25880.htm)，交给IP层。

　　常见的接口层协议有：

　　Ethernet 802.3、Token Ring 802.5、[X.25](http://baike.baidu.com/view/175390.htm)、[Frame relay](http://baike.baidu.com/view/284453.htm)、[HDLC](http://baike.baidu.com/view/89174.htm)、PPP ATM等。

### 网络层

　　负责相邻[计算机](http://baike.baidu.com/view/3314.htm)之间的通信。其功能包括三方面。

　　一、处理来自传输层的分组发送请求，收到请求后，将分组装入IP数据报，填充报头，选择去往信宿机的路径，然后将数据报发往适当的网络接口。

　　二、处理输入数据报：首先检查其合法性，然后进行寻径--假如该数据报已到达信宿机，则去掉报头，将剩下部分交给适当的传输协议；假如该数据报尚未到达信宿，则转发该数据报。

　　三、处理路径、流控、拥塞等问题。

　　网络层包括：IP(Internet Protocol）协议、ICMP(Internet Control Message Protocol)

　　控制报文协议、ARP(Address Resolution Protocol）地址转换协议、RARP(Reverse ARP)反向地址转换协议。

　　IP是网络层的核心，通过路由选择将下一条IP封装后交给接口层。IP数据报是无连接服务。

[ICMP](http://baike.baidu.com/view/30564.htm)是网络层的补充，可以回送报文。用来检测网络是否通畅。

[Ping](http://baike.baidu.com/view/709.htm)命令就是发送ICMP的[echo](http://baike.baidu.com/view/32757.htm)包，通过回送的echo relay进行网络测试。

[ARP](http://baike.baidu.com/view/32698.htm)是正向[地址解析协议](http://baike.baidu.com/view/149421.htm)，通过已知的IP，寻找对应主机的[MAC地址](http://baike.baidu.com/view/69334.htm)。

[RARP](http://baike.baidu.com/view/32772.htm)是反向地址解析协议，通过MAC地址确定IP地址。比如无盘工作站还有DHCP服务。

### 传输层

　　提供[应用程序](http://baike.baidu.com/view/330120.htm)间的通信。其功能包括：一、格式化信息流；二、提供可靠传输。为实现后者，传输层协议规定接收端必须发回确认，并且假如分组丢失，必须重新发送，即耳熟能详的“三次握手”过程，从而提供可靠的数据传输。

　　传输层协议主要是：传输控制协议TCP(Transmission Control Protocol）和用户数据报协议[UDP](http://baike.baidu.com/view/30509.htm)(User Datagram protocol）。

### 应用层

　　向用户提供一组常用的应用程序，比如[电子邮件](http://baike.baidu.com/view/1524.htm)、文件传输访问、[远程登录](http://baike.baidu.com/view/59099.htm)等。远程登录[TELNET](http://baike.baidu.com/view/44255.htm)使用TELNET协议提供在网络其它主机上注册的接口。TELNET会话提供了基于字符的虚拟终端。文件传输访问FTP使用[FTP协议](http://baike.baidu.com/view/1157060.htm)来提供网络内机器间的文件拷贝功能。

　　应用层协议主要包括如下几个：FTP、TELNET、DNS、SMTP、RIP、NFS、HTTP。

[FTP](http://baike.baidu.com/view/369.htm)(File Transfer Protocol）是文件传输协议，一般上传下载用FTP服务，数据端口是20H，控制端口是21H。

　　Telnet服务是用户远程登录服务，使用23H端口，使用明码传送，保密性差、简单方便。

[DNS](http://baike.baidu.com/view/22276.htm)(Domain Name Service）是域名解析服务，提供域名到IP地址之间的转换。

[SMTP](http://baike.baidu.com/view/5450.htm)(Simple Mail Transfer Protocol）是简单邮件传输协议，用来控制信件的发送、中转。

[RIP](http://baike.baidu.com/view/30530.htm) （Router Information Protocol）是路由信息协议，用于网络设备之间交换路由信息。

[NFS](http://baike.baidu.com/view/44349.htm) （Network File System）是网络文件系统，用于网络中不同主机间的文件共享。

[HTTP](http://baike.baidu.com/view/9472.htm)(Hypertext Transfer Protocol）是超文本传输协议，用于实现互联网中的WWW服务。

|  |  |  |
| --- | --- | --- |
| 总结 | | |
| **OSI中的层** | **功能** | **TCP/IP协议族** |
| 应用层 | 文件传输，电子邮件，文件服务，虚拟终端 | TFTP，HTTP，SNMP，FTP，SMTP，DNS，RIP，Telnet 等等 |
| 表示层 | 数据格式化，代码转换，数据加密 | 没有协议 |
| 会话层 | 解除或建立与别的接点的联系 | 没有协议 |
| 传输层 | 提供[端对端](http://baike.baidu.com/view/1145126.htm)的接口 | TCP，UDP |
| 网络层 | 为[数据包](http://baike.baidu.com/view/25880.htm)选择路由 | IP，ICMP，OSPF，EIGRP，IGMP，ARP，RARP |
| 数据链路层 | 传输有地址的帧以及错误检测功能 | SLIP，CSLIP，PPP，MTU，ARP，RARP |
| 物理层 | 以二进制数据形式在物理媒体上传输数据 | ISO2110，IEEE802，IEEE802.2 |

网络层中的协议主要有IP，ICMP，IGMP等，由于它包含了IP协议模块，所以它是所有基于TCP/IP协议网络的核心。在网络层中，IP模块完成大部分功能。ICMP和IGMP以及其他支持IP的协议帮助IP完成特定的任务，如传输差错控制信息以及主机/路由器之间的控制电文等。网络层掌管着网络中主机间的信息传输。

　　传输层上的主要协议是TCP和UDP。正如网络层控制着主机之间的数据传递，传输层控制着那些将要进入网络层的数据。两个协议就是它管理这些数据的两种方式：TCP是一个基于连接的协议；UDP则是面向无连接服务的管理方式的协议。

　　TCP/IP模型的主要缺点有：

　　第一，它在服务、接口与协议的区别上就不是很清楚。一个好的软件工程应该将功能与实现方法区分开来，TCP/IP恰恰没有很好地做到这点，就使得TCP/IP参考模型对于使用新的技术的指导意义是不够的。TCP/IP参考模型不适合于其他非TCP/IP协议簇。

　　第二，主机-网络层本身并不是实际的一层，它定义了网络层与数据链路层的接口。物理层与数据链路层的划分是必要和合理的，一个好的参考模型应该将它们区分开，而TCP/IP参考模型却没有做到这点。

## 数据格式

　　数据帧：帧头+IP数据包+帧尾 （帧头包括源和目标主机MAC地址及类型，帧尾是校验字）

　　IP数据包：IP头部+TCP数据信息（IP头包括源和目标主机IP地址、类型、生存期等）

　　TCP数据信息：TCP头部+实际数据 (TCP头包括源和目标主机[端口号](http://baike.baidu.com/view/642103.htm)、顺序号、确认号、校验字等）

## 产生背景

　　在阿帕网（ARPA）产生运作之初，通过接口信号处理机实现互联的电脑并不多，大部分电脑相互之间不兼容。在一台电脑上完成的工作，很难拿到另一台电脑上去用，想让硬件和软件都不一样的电脑联网，也有很多困难。当时美国的状况是，陆军用的电脑是DEC系列产品，海军用的电脑是Honeywell中标机器，空军用的是IBM公司中标的电脑，每一个军种的电脑在各自的系里都运行良好，但却有一个大弊病：不能共享资源。

　　当时科学家们提出这样一个理念：“所有电脑生来都是平等的。”为了让这些“生来平等”的电脑能够实现“资源共享”就得在这些系统的标准之上，建立一种大家共同都必须遵守的标准，这样才能让不同的电脑按照一定的规则进行“谈判”，并且在谈判之后能“握手”。

　　在确定今天[因特网](http://baike.baidu.com/view/1706.htm)各个电脑之间“谈判规则”过程中，最重要的人物当数瑟夫（Vinton G.Cerf）。正是他的努力，才使今天各种不同的电脑能按照协议上网互联。瑟夫也因此获得了与克莱因罗克（“因特网之父”）一样的美称“[互联网](http://baike.baidu.com/view/6825.htm)之父”。

　　瑟夫从小喜欢标新立异，坚强而又热情。中学读书时，就被允许使用加州大学洛杉矶分校的电脑，他认为“为电脑编程序是个非常激动人心的事，…只要把程序编好，就可以让电脑做任何事情。”1965年，瑟夫从斯坦福大学毕业到IBM的一家公司当系统[工程师](http://baike.baidu.com/view/25007.htm)，工作没多久，瑟夫就觉得知识不够用，于是到加州大学洛杉矶分校攻读博士，那时，正逢阿帕网的建立，“接口信号处理机”（IMP）的研试及网络测评中心的建立，瑟夫也成了著名科学家克莱因罗克手下的一位学生。瑟夫与另外三位年轻人（温菲尔德、克罗克、布雷登）参与了阿帕网的第一个节点的联接。此后不久，BBN公司对工作中各种情况发展有很强判断能力、被公认阿帕网建成作出巨大贡献的鲍伯·卡恩（Bob Kahn）也来到了加州大学洛杉矶分校。在那段日子里，往往是卡恩提出需要什么软件，而瑟夫则通宵达旦地把符合要求的软件给编出来，然后他们一起测试这些软件，直至能正常运行。

　　当时的主要格局是这样的，罗伯茨提出网络思想设计网络布局，卡恩设计阿帕网总体结构，克莱因罗克负责网络测评系统，还有众多的科学家、研究生参与研究、试验。69年9月阿帕网诞生、运行后，才发现各个IMP连接的时候，需要考虑用各种电脑都认可的信号来打开通信管道，数据通过后还要关闭通道。否则这些IMP不会知道什么时候应该接收信号，什么时候该结束，这就是我们现在所说的通信“协议”的概念。70年12月制定出来了最初的通信协议由卡恩开发、瑟夫参与的“网络控制协议”（NCP），但要真正建立一个共同的标准很不容易，72年10月国际电脑通信大会结束后，科学家们都在为此而努力。

　　“包切换”理论为网络之间的联接方式提供了理论基础。卡恩在自己研究的基础上，认识到只有深入理解各种操作系统的细节才能建立一种对各种操作系统普适的协议，73年卡恩请瑟夫一起考虑这个协议的各个细节，他们这次合作的结果产生了目前在开放系统下的所有网民和网管人员都在使用的“传输控制协议”（TCP，Transmission-Control Protocol）和“因特网协议”（IP，Internet Protocol）即TCP/IP协议。

　　通俗而言：TCP负责发现传输的问题，一有问题就发出信号，要求重新传输，直到所有数据安全正确地传输到目的地。而IP是给因特网的每一台电脑规定一个地址。1974年12月，卡恩、瑟夫的第一份TCP协议详细说明正式发表。当时美国国防部与三个科学家小组签定了完成TCP/IP的协议，结果由瑟夫领衔的小组捷足先登，首先制定出了通过详细定义的TCP/IP协议标准。当时作了一个试验，将信息包通过点对点的卫星网络，再通过陆地电缆，再通过卫星网络，再由地面传输，贯串欧洲和美国，经过各种电脑系统，全程9.4万公里竟然没有丢失一个数据位，远距离的可靠数据传输证明了TCP/IP协议的成功。

　　1983年1月1日，运行较长时期曾被人们习惯了的NCP被停止使用，TCP/IP协议作为因特网上所有主机间的共同协议，从此以后被作为一种必须遵守的规则被肯定和应用。

## 开发过程

　　在构建了[阿帕网](http://baike.baidu.com/view/108095.htm)先驱之后，DARPA开始了其他数据传输技术的研究。NCP诞生后两年，1972年，[罗伯特·卡恩](http://baike.baidu.com/view/2020429.htm)（Robert E. Kahn）被DARPA的信息技术处理办公室雇佣，在那里他研究卫星数据包网络和地面无线数据包网络，并且意识到能够在它们之间沟通的价值。在1973年春天，已有的ARPANET网络控制程序（NCP）协议的开发者[文顿·瑟夫](http://baike.baidu.com/view/5129002.htm)（Vinton Cerf）加入到卡恩为ARPANET设计下一代协议而开发开放互连模型的工作中。

　　到了[1973年](http://baike.baidu.com/view/286653.htm)夏天，卡恩和瑟夫很快就开发出了一个基本的改进形式，其中[网络协议](http://baike.baidu.com/view/16603.htm)之间的不同通过使用一个公用互联网络协议而隐藏起来，并且可靠性由主机保证而不是像ARPANET那样由网络保证。（瑟夫称赞Hubert Zimmerman和Louis Pouzin（CYCLADES网络的设计者）在这个设计上发挥了重要影响。）

　　由于网络的作用减少到最小的程度，就有可能将任何网络连接到一起，而不用管它们不同的特点，这样就解决了卡恩最初的问题。（一个流行的说法提到瑟夫和卡恩工作的最终产品[TCP/IP](http://baike.baidu.com/view/7729.htm)将在运行“两个罐子和一根弦”上，实际上它已经用在信鸽上。一个称为*网关*（后来改为[*路由器*](http://baike.baidu.com/view/1360.htm)以免与[*网关*](http://baike.baidu.com/view/807.htm)混淆）的计算机为每个网络提供一个接口并且在它们之间来回传输[数据包](http://baike.baidu.com/view/25880.htm)。

　　这个设计思想更细的形式由瑟夫在斯坦福的网络研究组的1973年–[1974年](http://baike.baidu.com/view/286308.htm)期间开发出来。（处于同一时期的诞生了PARC通用包协议组的施乐PARC早期网络研究工作也有重要的技术影响；人们在两者之间摇摆不定。）

　　DARPA于是与BBN、斯坦福和伦敦大学签署了协议开发不同硬件平台上协议的运行版本。有四个版本被开发出来——TCP v1、TCP v2、在1978年春天分成TCP v3和IP v3的版本，后来就是稳定的TCP/IP v4——目前因特网仍然使用的标准协议。

[1975年](http://baike.baidu.com/view/286651.htm)，两个网络之间的TCP/IP通信在斯坦福和伦敦大学（UCL）之间进行了测试。[1977年](http://baike.baidu.com/view/286649.htm" \t "_blank)11月，三个网络之间的TCP/IP测试在美国、英国和挪威之间进行。在[1978年](http://baike.baidu.com/view/286309.htm)到1983年间，其他一些TCP/IP原型在多个研究中心之间开发出来。ARPANET完全转换到TCP/IP在1983年1月1日发生。[1]

　　1984年，美国国防部将TCP/IP作为所有[计算机网络](http://baike.baidu.com/view/25482.htm)的标准。1985年，因特网架构理事会举行了一个三天有250家厂商代表参加的关于计算产业使用TCP/IP的工作会议，帮助协议的推广并且引领它日渐增长的商业应用。

[2005年](http://baike.baidu.com/view/286352.htm)[9月9日](http://baike.baidu.com/view/477852.htm)卡恩和瑟夫由于他们对于美国文化做出的卓越贡献被授予[总统自由勋章](http://baike.baidu.com/view/1984059.htm)。

## 运作机制

### 1.IP

　　IP层接收由更低层（网络接口层例如[以太网](http://baike.baidu.com/view/848.htm)设备驱动程序）发来的数据包，并把该数据包发送到更高层---TCP或UDP层；相反，IP层也把从TCP或UDP层接收来的数据包传送到更低层。IP数据包是不可靠的，因为IP并没有做任何事情来确认数据包是按顺序发送的或者没有被破坏。IP数据包中含有发送它的主机的地址（源地址）和接收它的主机的地址（目的地址）。

　　高层的TCP和UDP服务在接收数据包时，通常假设包中的源地址是有效的。也可以这样说，[IP地址](http://baike.baidu.com/view/3930.htm)形成了许多服务的认证基础，这些服务相信数据包是从一个有效的主机发送来的。IP确认包含一个选项，叫作IP source routing，可以用来指定一条源地址和目的地址之间的直接路径。对于一些TCP和UDP的服务来说，使用了该选项的IP包好像是从路径上的最后一个系统传递过来的，而不是来自于它的真实地点。这个选项是为了测试而存在的，说明了它可以被用来欺骗系统来进行平常是被禁止的连接。那么，许多依靠IP源地址做确认的服务将产生问题并且会被非法入侵。

### 2.TCP

　　TCP是面向连接的通信协议，通过[三次握手](http://baike.baidu.com/view/1003841.htm)建立连接，通讯完成时要拆除连接，由于TCP是面向连接的所以只能用于点对点的通讯。

　　TCP提供的是一种可靠的[数据流](http://baike.baidu.com/view/166248.htm)服务，采用“带重传的肯定确认”技术来实现传输的可靠性。TCP还采用一种称为“滑动窗口”的方式进行流量控制，所谓窗口实际表示接收能力，用以限制发送方的发送速度。

　　如果IP数据包中有已经封好的TCP数据包，那么IP将把它们向‘上’传送到TCP层。TCP将包排序并进行错误检查，同时实现虚电路间的连接。TCP数据包中包括序号和确认，所以未按照顺序收到的包可以被排序，而损坏的包可以被重传。

　　TCP将它的信息送到更高层的应用程序，例如Telnet的服务程序和客户程序。应用程序轮流将信息送回TCP层，TCP层便将它们向下传送到IP层，设备驱动程序和物理介质，最后到接收方。

　　面向连接的服务（例如Telnet、FTP、rlogin、X Windows和SMTP）需要高度的可靠性，所以它们使用了TCP。DNS在某些情况下使用TCP（发送和接收[域名](http://baike.baidu.com/view/43.htm)数据库），但使用UDP传送有关单个主机的信息。

### 3.UDP

　　UDP是面向无连接的通讯协议，UDP数据包括目的端口号和源端口号信息，由于通讯不需要连接，所以可以实现广播发送。

　　UDP通讯时不需要接收方确认，属于不可靠的传输，可能会出丢包现象，实际应用中要求在程序员编程验证。

　　UDP与TCP位于同一层，但它不管数据包的顺序、错误或重发。因此，UDP不被应用于那些使用虚电路的面向连接的服务，UDP主要用于那些面向查询---应答的服务，例如NFS。相对于FTP或Telnet，这些服务需要交换的信息量较小。使用UDP的服务包括NTP（网络时间协议）和DNS（DNS也使用TCP）。

　　欺骗UDP包比欺骗TCP包更容易，因为UDP没有建立初始化连接（也可以称为握手）（因为在两个系统间没有虚电路），也就是说，与UDP相关的服务面临着更大的危险。

### 4.ICMP

　　ICMP与IP位于同一层，它被用来传送IP的的控制信息。它主要是用来提供有关通向目的地址的路径信息。ICMP的‘Redirect’信息通知主机通向其他系统的更准确的路径，而‘Unreachable’信息则指出路径有问题。另外，如果路径不可用了，ICMP可以使TCP连接‘体面地’终止。PING是最常用的基于ICMP的服务。

## 通讯端口

　　TCP和UDP服务通常有一个客户/[服务器](http://baike.baidu.com/view/899.htm)的关系，例如，一个Telnet服务进程开始在系统上处于空闲状态，等待着连接。用户使用Telnet客户程序与服务进程建立一个连接。客户程序向服务进程写入信息，服务进程读出信息并发出响应，客户程序读出响应并向用户报告。因而，这个连接是双工的，可以用来进行读写。

　　两个系统间的多重Telnet连接是如何相互确认并协调一致呢？TCP或UDP连接唯一地使用每个信息中的如下四项进行确认：

　　源IP地址 发送包的IP地址。

　　目的IP地址 接收包的IP地址。

　　源端口 源系统上的连接的端口。

　　目的端口 目的系统上的连接的端口。

　　端口是一个软件结构，被客户程序或服务进程用来发送和接收信息。一个端口对应一个16比特的数。服务进程通常使用一个固定的端口，例如，SMTP使用25、Xwindows使用6000。这些[端口号](http://baike.baidu.com/view/642103.htm)是‘广为人知’的，因为在建立与特定的主机或服务的连接时，需要这些地址和目的地址进行通讯。

## IP地址

　　在Internet上连接的所有计算机，从大型机到微型计算机都是以独立的身份出现，我们称它为主机。为了实现各主机间的通信，每台主机都必须有一个唯一的[网络地址](http://baike.baidu.com/view/547479.htm)。就好像每一个住宅都有唯一的门牌一样，才不至于在传输资料时出现混乱。

　　Internet的网络地址是指连入Internet网络的计算机的地址编号。所以，在Internet网络中，网络地址唯一地标识一台计算机。

　　我们都已经知道，Internet是由几千万台计算机互相连接而成的。而我们要确认网络上的每一台计算机，靠的就是能唯一标识该计算机的网络地址，这个地址就叫做IP（Internet Protocol的简写）地址，即用Internet协议语言表示的地址。

　　目前，在Internet里，IP地址是一个32位的二进制地址，为了便于记忆，将它们分为4组，每组8位，由小数点分开，用四个字节来表示，而且，用点分开的每个字节的数值范围是0~255，如202.116.0.1，这种书写方法叫做点数表示法。

### 地址分类

　　IP地址可确认网络中的任何一个网络和计算机，而要识别其它网络或其中的计算机，则是根据这些IP地址的分类来确定的。一般将IP地址按节点计算机所在网络规模的大小分为A，B，C三类，默认的网络屏蔽是根据IP地址中的第一个字段确定的。

　　1． A类地址

　　A类地址的表示范围为：1.0.0.1~126.255.255.255，默认网络屏蔽为：255.0.0.0；A类地址分配给规模特别大的网络使用。A类网络用第一组数字表示网络本身的地址，后面三组数字作为连接于网络上的主机的地址。分配给具有大量主机（直接个人用户）而[局域网](http://baike.baidu.com/view/788.htm)络个数较少的大型网络。例如IBM公司的网络。

　　127.0.0.0到127.255.255.255是保留地址，用做循环测试用的。

　　0.0.0.0到0.255.255.255也是保留地址，用做表示所有的IP地址。

　　一个[A类IP地址](http://baike.baidu.com/view/15408.htm)由1字节（每个字节是8位）的网络地址和3个字节主机地址组成，网络地址的最高位必须是“0”，即第一段数字范围为1～127。每个A类地址理论上可连接16777214<256\*256\*256-2>；台主机（-2是因为主机中要用去一个网络号和一个广播号），Internet有126个可用的A类地址。A类地址适用于有大量主机的大型网络。

　　2． B类地址

　　B类地址的表示范围为：128.0.0.1~191.255.255.255，默认网络屏蔽为：255.255.0.0；B类地址分配给一般的中型网络。B类网络用第一、二组数字表示网络的地址，后面两组数字代表网络上的主机地址。

　　169.254.0.0到169.254.255.255是保留地址。如果你的[IP地址](http://baike.baidu.com/view/3930.htm)是自动获取IP地址，而你在网络上又没有找到可用的DHCP服务器，这时你将会从169.254.0.0到169.254.255.255中临时获得一个IP地址。

　　一个B类IP地址由2个字节的网络地址和2个字节的主机地址组成，网络地址的最高位必须是“10”，即第一段数字范围为128～191。每个B类地址可连接65534(2^16-2，因为主机号的各位不能同时为0,1）台主机，Internet有16383(2^14-1）个B类地址（因为B类网络地址128.0.0.0是不指派的，而可以指派的最小地址为128.1.0.0[COME06]）。

　　3． C类地址

　　C类地址的表示范围为：192.0.0.1~223.255.255.255，默认网络屏蔽为：255.255.255.0；C类地址分配给小型网络，如一般的局域网，它可连接的主机数量是最少的，采用把所属的用户分为若干的网段进行管理。C类网络用前三组数字表示网络的地址，最后一组数字作为网络上的主机地址。

　　一个C类地址是由3个字节的网络地址和1个字节的主机地址组成，网络地址的最高位必须是“110”，即第一段数字范围为192～223。每个C类地址可连接254台主机，Internet有2097152个C类地址段（32\*256\*256），有532676608个地址（32\*256\*256\*254）。

　　RFC 1918留出了3块IP地址空间（1个A类地址段，16个B类地址段，256个C类地址段）作为私有的内部使用的地址。在这个范围内的IP地址不能被路由到Internet骨干网上；Internet[路由器](http://baike.baidu.com/view/1360.htm)将丢弃该私有地址。

　　IP地址类别　RFC 1918内部地址范围

　　A类　10．0.0.0到10.255.255.255

　　B类　172.16.0.0到172.31.255.255

　　C类　192.168.0.0到192.168.255.255

　　使用私有地址将网络连至Internet，需要将私有地址转换为公有地址。这个转换过程称为网络地址转换（Network Address Translation，NAT），通常使用路由器来执行NAT转换。

　　实际上，还存在着D类地址和E类地址。但这两类地址用途比较特殊，在这里只是简单介绍一下：

　　D类地址不分网络地址和主机地址，它的第1个字节的前四位固定为1110。D类地址范围：224.0.0.1到239.255.255.254。D类地址用于多点播送。D类地址称为[组播地址](http://baike.baidu.com/view/1871353.htm)（或称多播地址），供特殊协议向选定的节点发送信息时用。

　　E类地址保留给将来使用。

　　连接到Internet上的每台计算机，不论其IP地址属于哪类都与网络中的其它计算机处于平等地位，因为只有IP地址才是区别计算机的唯一标识。所以，以上IP地址的分类只适用于网络分类。

　　在Internet中，一台计算机可以有一个或多个IP地址，就像一个人可以有多个通信地址一样，但两台或多台计算机却不能共享一个IP地址。如果有两台计算机的IP地址相同，则会引起异常现象，无论哪台计算机都将无法正常工作。

　　顺便提一下几类特殊的IP地址：

　　1． 广播地址目的端为给定网络上的所有主机，一般主机段为全1

　　2． 单播地址目的端为指定网络上的单个主机地址

　　3． 组播地址目的端为同一组内的所有主机地址

　　4． 环回地址[127.0.0.1](http://baike.baidu.com/view/971216.htm)在环回测试和广播测试时会使用

### 网关地址

　　若要使两个完全不同的网络（异构网）连接在一起，一般使用网关，在Internet中两个网络也要通过一台称为网关的计算机实现互联。这台计算机能根据用户通信目标计算机的IP地址，决定是否将用户发出的信息送出本地网络，同时，它还将外界发送给属于本地网络计算机的信息接收过来，它是一个网络与另一个网络相联的通道。为了使TCP/IP协议能够寻址，该通道被赋予一个IP地址，这个IP地址称为网关地址。

## 注意事项

　　内部地址和外部地址在局域网的IP地址分配中，并没有区别，都可以使用。

　　在局域网的IP地址分配中，子网屏蔽的“1”部分只要和对应的IP地址分类规定的前几个二进制数一致即可。

## 子网划分

　　若公司不上Internet，那一定不会烦恼IP地址的问题，因为可以任意使用所有的IP地址，不管是A类或是B类，这个时候不会想到要用子网，但若是上Internet那IP地址便弥足珍贵了，目前全球一阵Internet热,IP地址已经愈来愈少了，而所申请的IP地址目前也趋饱和，而且只有经申请的IP地址能在Internet使用，但对某些公司只能申请到一个C类的IP地址，但又有多个点需要使用，那这时便需要使用到子网，这就需要考虑子网的划分，下面简介子网的原理及如何规划。

### 子网掩码

　　设定任何网络上的任何设备不管是主机、个人电脑、路由器等皆需要设定IP地址，而跟随着IP地址的是所谓的子网掩码（NetMask,Subnet Mask），这个子网掩码主要的目的是由IP地址中也能获得网络编码，也就是说IP地址和子网掩码合作而得到网络编码，如下所示：

　　IP地址

　　192.10.10.6 11000000.00001010.00001010.00000110

　　子网掩码

　　255.255.255.0 11111111.11111111.11111111.00000000

　　AND

　　-------------------------------------------------------------------

　　Network Number

　　192.10.10.0 11000000.00001010.00001010.00000000

　　子网掩码有所谓的默认值，如下所示

　　类 IP地址范围 子网掩码

　　A　1．0.0.0-126.255.255.255 255.0.0.0

　　B　128.0.0.0-191.255.255.255 255.255.0.0

　　C　192.0.0.0-223.255.255.255 255.255.255.0

　　在预设的子网掩码（Net Mask）都只有255的值，在谈到子网掩码（Subnet Mask）时这个值便不一定是255了。在完整一组C类地址中如203.67.10.0-203.67.10.255 子网掩码255.255.255.0,203.67.10.0称之网络编码（Network Number，将IP 地址和子网掩码作和），而203.67.10.255是广播的IP地址，所以这两者皆不能使用，实际只能使用203.67.10.1--203.67.10.254等254个IP地址，这是以255.255.255.0作子网掩码的结果，而所谓Subnet Msk尚可将整组C类地址分成数组网络编码，这要在子网掩码上作手脚，若是要将整组C类地址分成2个网络编码那子网掩码设定为255.255.255.128，若是要将整组C类分成8组网络编码则子网掩码要为255.255.255.224，这是怎么来的，由以上知道网络编码是由IP地址和子网掩码作AND而来的，而且将子网掩码以二进制表示法知道是1的会保留，而为0的去掉

　　192.10.10.193--11000000.00001010.00001010.11000001

　　255.255.255.0--11111111.11111111.11111111.00000000

　　--------------------------------------------------------------

　　192.10.10.0--11000000.00001010.00001010.00000000

　　以上是以255.255.255.0为子网掩码的结果，网络编码是192.10.10.0，若是使用255.255.255.224作子网掩码结果便有所不同

　　192.10.10.193--11000000.00001010.00001010.11000001

　　255.255.255.224--11111111.11111111.11111111.11100000

　　--------------------------------------------------------------

　　192.10.10.192--11000000.00001010.00001010.11000000

　　此时网络编码变成了192.10.10.192，这便是子网。那要如何决定所使用的子网掩码，255.255.255.224以二进制表示法为11111111.11111111.11111111.11100000，变化是在最后一组，11100000便是224，以三个位（Bit）可表示2的3次方便是8个网络编码

　　子网掩码二进制表示法可分几个网络

　　255.255.255.011111111.11111111.11111111.000000001

　　255.255.255.128

　　11111111.11111111.11111111.100000002

　　255.255.255.192

　　11111111.11111111.11111111.110000004

　　255.255.255.224

　　11111111.11111111.11111111.111000008

　　255.255.255.240

　　11111111.11111111.11111111.1111000016

　　255.255.255.248

　　11111111.11111111.11111111.1111100032

　　255.255.255.252

　　11111111.11111111.11111111.1111110064

　　以下使用255.255.255.224将C类地址203.67.10.0分成8组网络编码，各个网络编码及其广播IP地址及可使用之IP地址序号网络编码广播可使用之IP地址

　　（1）203.67.10.0--203.67.10.31

　　203.67.10.1--203.67.10.30

　　（2）203.67.10.32--203.67.10.63

　　203.67.10.33--203.67.10.62

　　（3）203.67.10.64--203.67.10.95

　　203.67.10.65--203.67.10.94

　　（4）203.67.10.96--203.67.10.127

　　203.67.10.97--203.67.10.126

　　（5）203.67.10.128--203.67.10.159

　　203.67.10.129--203.67.10.158

　　（6）203.67.10.160--203.67.10.191

　　203.67.10.161--203.67.10.190

　　（7）203.67.10.192--203.67.10.223

　　203.67.10.193--203.67.10.222

　　（8）203.67.10.224--203.67.10.255

　　203.67.10.225--203.67.10.254

　　可验证所使用的IP地址是否如上表所示

　　203.67.10.115--11001011.01000011.00001010.01110011

　　255.255.255.224--11111111.11111111.11111111.11100000

　　--------------------------------------------------------------

　　203.67.10.96--11001011.01000011.00001010.01100000

　　203.67.10.55--11001011.01000011.00001010.00110111

　　255.255.255.224--11111111.11111111.11111111.11100000

　　--------------------------------------------------------------

　　203.67.10.32--11001011.01000011.00001010.00100000

　　其它的子网掩码所分成的网络编码可自行以上述方法自行推演出来。

### 子网作用

　　使用子网是要解决只有一组C类地址但需要数个网络编码的问题，并不是解决IP地址不够用的问题，因为使用子网反而能使用的IP地址会变少，子网通常是使用在跨地域的网络互联之中，两者之间使用路由器连线，同时也上Internet，但只申请到一组C 类IP地址，过路由又需不同的网络，所以此时就必须使用到子网，当然二网络间也可以远程桥接（Remote Bridge，字面翻译）连接，那便没有使用子网的问题。

## 概念区分

### 协议阐述IP申请的术语

　　IP申请的术语:

　　网络地址：在申请IP地址或是阐述TCP/IP协议的IP地址分类时，用到这个术语。它表示IP地址的代码序列中不可根据需要改变的部分。

　　主机地址：在申请IP地址或是阐述TCP/IP协议的IP地址分类时，用到这个术语。它表示IP地址的代码序列中能够根据需要来改变的部分。

　　子网屏蔽：在阐述TCP/IP协议的IP地址分类时，用到这个术语。在申请IP地址时，由它表示所申请到的IP地址的网络地址和主机地址。

### 子网划分

　　内网搭建的术语:

　　网络编码（网络号）：经过子网划分后，子网掩码序列中“1”对应的IP地址部分。一个网络编码，对应一个网域（或网段）。包括申请到的网络地址的全部和主机地址的部分。

　　主机编码（主机号）：经过子网划分后，子网掩码序列中“0”对应的IP地址部分。一个主机编码，对应一个网域（或网段）的一台计算机。包括申请到主机地址的部分。

　　子网掩码：用于子网划分，它将能够改变的主机地址分为主机编码和网络编码的一部分。同时，它将网络地址全部确定为网络编码。

## 协议测试

　　全面的测试应包括局域网和互联网两个方面，因此应从局域网和互联网两个方面测试，以下是在实际工作中利用命令行测试TCP/IP配置步骤：

　　1． 单击“开始”/“运行”，输入CMD按回车，打开命令提示符窗口。

　　2．首先检查IP地址、子网掩码、[默认网关](http://baike.baidu.com/view/119302.htm)、DNS服务器地址是否正确，输入命令ipconfig /all，按回车。此时显示了你的网络配置，观查是否正确。

　　3．输入ping 127.0.0.1，观查网卡是否能转发数据，如果出现“Request timed out”，表明配置差错或网络有问题。

　　4．Ping一个互联网地址，看是否有数据包传回，以验证与互联网的连接性。

　　5． Ping 一个局域网地址，观查与它的连通性。

　　6．用nslookup测试DNS解析是否正确，输入如nslookup ，查看是否能解析。

　　如果你的计算机通过了全部测试，则说明网络正常，否则网络可能有不同程度的问题。在此不展开详述。不过，要注意，在使用 ping命令时，有些公司会在其主机设置丢弃ICMP数据包，造成你的ping命令无法正常返回数据包，不防换个网站试试。

## 主要特点

　　（1）TCP/IP协议不依赖于任何特定的计算机硬件或操作系统，提供开放的协议标准，即使不考虑Internet，TCP/IP协议也获得了广泛的支持。所以TCP/IP协议成为一种联合各种硬件和软件的实用系统。

　　（2）TCP/IP协议并不依赖于特定的网络传输硬件，所以TCP/IP协议能够集成各种各样的网络。用户能够使用以太网（Ethernet）、令牌环网（Token Ring Network）、拨号线路（Dial-up line）、X.25网以及所有的网络传输硬件。

　　（3）统一的网络地址分配方案，使得整个TCP/IP设备在网中都具有惟一的地址

　　（4）标准化的高层协议，可以提供多种可靠的用户服务。

## 协议优势

　　在长期的发展过程中，IP逐渐取代其他网络。这里是一个简单的解释。IP传输通用数据。数据能够用于任何目的，并且能够很轻易地取代以前由专有数据网络传输的数据。下面是一个普通的过程：

　　一个专有的网络开发出来用于特定目的。如果它工作很好，用户将接受它。

　　为了便利提供IP服务，经常用于访问电子邮件或者聊天，通常以某种方式通过专有网络隧道实现。隧道方式最初可能非常没有效率，因为电子邮件和聊天只需要很低的带宽。

　　通过一点点的投资IP 基础设施逐渐在专有数据网络周边出现。

　　用IP取代专有服务的需求出现，经常是一个用户要求。

　　IP替代品过程遍布整个因特网，这使IP替代品比最初的专有网络更加有价值（由于网络效应）。

　　专有网络受到压制。许多用户开始维护使用IP替代品的复制品。

　　IP包的间接开销很小，少于1%，这样在成本上非常有竞争性。人们开发了一种能够将IP带到专有网络上的大部分用户的不昂贵的传输媒介。

　　大多数用户为了削减开销，专有网络被取消。

## 协议重置

　　如果需要重新安装 TCP/IP 以使TCP/IP 堆栈恢复为原始状态。可以使用NetShell 实用程序重置TCP/IP 堆栈，使其恢复到初次安装操作系统时的状态。具体操作如下：

　　1．单击 开始--> 运行，输入"CMD" 后单击"确定";

　　2．在命令行模式输入命令

　　netsh int ip reset C:\resetlog.txt

　　（其中，Resetlog.txt记录命令结果的日志文件，一定要指定，这里指定了Resetlog.txt 日志文件及完整路径。）

　　运行结果可以查看C:\resetlog.txt （咨询中可根据用户实际操作情况提供）

　　运行此命令的结果与删除并重新安装TCP/IP 协议的效果相同。

**注意**

　　本操作具有一定的风险性，请在操作前备份重要数据，并根据操作熟练度酌情使用。

## 版本更新

### IPV4

　　IPv4，是[互联网协议](http://baike.baidu.com/view/1087323.htm)（Internet Protocol，IP）的第四版，也是第一个被广泛使用，构成现今[互联网技术](http://baike.baidu.com/view/1444237.htm)的基石的协议。1981年Jon Postel 在RFC791中定义了IP，Ipv4可以运行在各种各样的底层网络上，比如端对端的串行数据链路（PPP协议和[SLIP协议](http://baike.baidu.com/view/525525.htm)) ，卫星链路等等。局域网中最常用的是以太网。

　　传统的TCP/IP协议基于[IPV4](http://baike.baidu.com/view/21992.htm)属于[第二代互联网](http://baike.baidu.com/view/533978.htm)技术，核心技术属于美国。它的最大问题是网络地址资源有限，从理论上讲，编址1600万个网络、40亿台主机。但采用A、B、C三类编址方式后，可用的网络地址和主机地址的数目大打折扣，以至目前的IP地址已经枯竭。其中[北美](http://baike.baidu.com/view/595143.htm)占有3/4，约30亿个，而人口最多的亚洲只有不到4亿个，[中国](http://baike.baidu.com/view/61891.htm)截止2010年6月IPv4地址数量达到2.5亿，落后于4.2亿网民的需求。虽然用[动态IP](http://baike.baidu.com/view/536227.htm)及Nat地址转换等技术实现了一些缓冲，但IPV4地址枯竭已经成为不争的事实。在此，专家提出IPV6的互联网技术，也正在推行，但IPV4的使用过度到IPV6需要很长的一段过度期。目前中国主要用的就是ip4，在win7中已近有了ipv6的协议不过对于中国的用户们来说可能很久以后才会用到吧。

　　传统的TCP/IP协议基于电话宽带以及[以太网](http://baike.baidu.com/view/848.htm)的电器特性而制定的，其分包原则与检验占用了数据包很大的一部分比例造成了传输效率低，现在网络正向着全[光纤网络](http://baike.baidu.com/view/2058721.htm)和超高速以太网方向发展，TCP/IP协议不能满足其发展需要。

　　1983年TCP/IP协议被[ARPAnet](http://baike.baidu.com/view/196838.htm)采用，直至发展到后来的互联网。那时只有几百台计算机互相联网。到1989年联网计算机数量突破10万台，并且同年出现了1.5Mbit/s的[骨干网](http://baike.baidu.com/view/1140231.htm)。因为[IANA](http://baike.baidu.com/view/155085.htm)把大片的[地址空间](http://baike.baidu.com/view/1507129.htm)分配给了一些公司和研究机构，[90年代](http://baike.baidu.com/view/1392115.htm)初就有人担心10年内IP地址空间就会不够用，并由此导致了IPv6 的开发。

### IPv6

　　IPv6是Internet Protocol Version 6的缩写，其中Internet Protocol译为“[互联网协议](http://baike.baidu.com/view/1087323.htm)”。IPv6是IETF（互联网工程任务组，Internet Engineering Task Force）设计的用于替代现行版本IP协议（IPv4）的下一代IP协议。

　　与IPV4相比，IPV6具有以下几个优势：

　　一、IPv6具有更大的地址空间。IPv4中规定IP地址长度为32，即有2^32-1（符号^表示升幂，下同）个地址；而IPv6中IP地址的长度为128，即有2^128-1个地址。

　　二、IPv6使用更小的[路由表](http://baike.baidu.com/view/149989.htm)。IPv6的地址分配一开始就遵循聚类（Aggregation）的原则，这使得路由器能在路由表中用一条记录（Entry）表示一片子网，大大减小了路由器中路由表的长度，提高了路由器转发数据包的速度。

　　三、IPv6增加了增强的组播（Multicast）支持以及对流的支持（Flow Control），这使得网络上的多媒体应用有了长足发展的机会，为[服务质量](http://baike.baidu.com/view/522662.htm)（QoS，Quality of Service）控制提供了良好的网络平台。

　　四、IPv6加入了对自动[配置](http://baike.baidu.com/view/132733.htm)（Auto Configuration）的支持。这是对[DHCP](http://baike.baidu.com/view/7992.htm)协议的改进和扩展，使得网络（尤其是局域网）的管理更加方便和快捷。

　　五、IPv6具有更高的安全性。在使用IPv6网络中用户可以对网络层的数据进行加密并对IP报文进行校验，极大的增强了网络的安全性。

扩展阅读：

* 1

一个局域网某台主机IP地址为176.68.160.12，使用22位作为网络地址，那么该局域网的子网掩码为（255.255.252.0）

* 2

最多可以连接的主机数为（1022）

# 以太网，万维网，因特网，互联网

互联网、因特网、万维网三者的关系是：互联网 包含 因特网 包含 万维网。

凡是能彼此通信的设备组成的网络就叫互联网。所以，即使仅有两台机器，不论用何种技术使其彼此通信，也叫互联网。国际标准的互联网写法是internet，字母i一定要小写！夸国性的超大互联网不仅有因特网，还有惠多网。

因特网是互联网的一种。因特网可不是仅有两台机器组成的互联网，它是由上千万台设备组成的互联网。因特网使用TCP/IP协议让不同的设备可以彼此通信。但使用TCP/IP协议的网络并不一定是因特网，一个局域网也可以使用TCP/IP协议。判断自己是否接入的是因特网，首先是看自己电脑是否安装了 TCP/IP协议，其次看是否拥有一个公网地址(所谓公网地址，就是所有私网地址以外的地址)。国际标准的因特网写法是Internet，字母I一定要大写！

因特网是基于TCP/IP协议实现的，TCP/IP协议由很多协议组成，不同类型的协议又被放在不同的层，其中，位于应用层的协议就有很多，比如FTP、SMTP、HTTP。只要应用层使用的是HTTP协议，就称为万维网(World Wide Web)。之所以在浏览器里输入<http://www.aorb.org>时，能看见矛盾网提供的网页，就是因为您的个人浏览器和矛盾网的服务器之间使用的是HTTP协议在交流。

只所以区分这三者，是因为很多书籍里行文不按以上标准行事，使读者容易产生困惑。

# [中缀表达式转后缀表达式](http://wawlian.iteye.com/blog/1266397)

我们在数学中常见的计算式，例如2+（3\*4）叫做中缀表达式。表达式中涉及到了多个运算符，而运算符之间是有优先级的。计算机在计算并且处理这种表达式时，需要将中缀表达式转换成后缀表达式，然后再进行计算。

中缀表达式转后缀表达式遵循以下原则：

1.遇到操作数，直接输出；

2.栈为空时，遇到运算符，入栈；

3.遇到左括号，将其入栈；

4.遇到右括号，执行出栈操作，并将出栈的元素输出，直到弹出栈的是左括号，左括号不输出；

5.遇到其他运算符'+''-''\*''/'时，弹出所有优先级大于或等于该运算符的栈顶元素，然后将该运算符入栈；

6.最终将栈中的元素依次出栈，输出。

经过上面的步骤，得到的输出既是转换得到的后缀表达式。

# [二叉树的深度优先遍历、广度优先遍历和非递归遍历](http://www.cnblogs.com/way_testlife/archive/2010/10/07/1845264.html)

二叉树的遍历：

D：访问根结点，L：遍历根结点的左子树，R：遍历根结点的右子树。

给定一棵二叉树的前序遍历序列和中序遍历序列可以惟一确定一棵二叉树。

二叉树的深度优先遍历的非递归的通用做法是采用栈，广度优先遍历的非递归的通用做法是采用队列。

**深度优先遍历二叉树。**

1. 中序遍历（LDR）的递归算法：

若二叉树为空，则算法结束；否则：

中序遍历根结点的左子树；

访问根结点；

中序遍历根结点的右子树。

2. 前序遍历（DLR）的递归算法：

若二叉树为空，则算法结束，否则：

访问根结点；

前序遍历根结点的左子树；

前序遍历根结点的右子树。

3. 后序遍历（LRD）的递归算法：

若二叉树为空，则算法结束，否则：

后序遍历根结点的左子树；

后序遍历根结点的右子树；

访问根结点。

**广度优先遍历二叉树。**

广度优先周游二叉树(层序遍历)是用队列来实现的，从二叉树的第一层（根结点）开始，自上至下逐层遍历；在同一层中，按照从左到右的顺序对结点逐一访问。

按照从根结点至叶结点、从左子树至右子树的次序访问二叉树的结点。算法：

1初始化一个队列，并把根结点入列队；

2当队列为非空时，循环执行步骤3到步骤5，否则执行6；

3出队列取得一个结点，访问该结点；

4若该结点的左子树为非空，则将该结点的左子树入队列；

5若该结点的右子树为非空，则将该结点的右子树入队列；

6结束。

**非递归深度优先遍历二叉树。**

栈是实现递归的最常用的结构，利用一个栈来记下尚待遍历的结点或子树，以备以后访问，可以将递归的深度优先遍历改为非递归的算法。

1. 非递归前序遍历：遇到一个结点，就访问该结点，并把此结点推入栈中，然后下降去遍历它的左子树。遍历完它的左子树后，从栈顶托出这个结点，并按照它的右链接指示的地址再去遍历该结点的右子树结构。

2. 非递归中序遍历：遇到一个结点，就把它推入栈中，并去遍历它的左子树。遍历完左子树后，从栈顶托出这个结点并访问之，然后按照它的右链接指示的地址再去遍历该结点的右子树。

3. 非递归后序遍历：遇到一个结点，把它推入栈中，遍历它的左子树。遍历结束后，还不能马上访问处于栈顶的该结点，而是要再按照它的右链接结构指示的地址去遍历该结点的右子树。遍历遍右子树后才能从栈顶托出该结点并访问之。另外，需要给栈中的每个元素加上一个特征位，以便当从栈顶托出一个结点时区别是从栈顶元素左边回来的(则要继续遍历右子树)，还是从右边回来的(该结点的左、右子树均已周游)。特征为Left表示已进入该结点的左子树，将从左边回来；特征为Right表示已进入该结点的右子树，将从右边回来。

4. 简洁的非递归前序遍历：遇到一个结点，就访问该结点，并把此结点的非空右结点推入栈中，然后下降去遍历它的左子树。遍历完左子树后，从栈顶托出一个结点，并按照它的右链接指示的地址再去遍历该结点的右子树结构。

----------------------------------------------------------------------

图的深度优先搜索法是树的先根遍历的推广，它的基本思想是：从图G的某个顶点v0出发，访问v0，然后选择一个与v0相邻且没被访问过的顶点vi访问，再从vi出发选择一个与vi相邻且未被访问的顶点vj进行访问，依次继续。如果当前被访问过的顶点的所有邻接顶点都已被访问，则退回到已被访问的顶点序列中最后一个拥有未被访问的相邻顶点的顶点w，从w出发按同样的方法向前遍历，直到图中所有顶点都被访问。  
图的广度优先搜索是树的按层次遍历的推广，它的基本思想是：首先访问初始点vi，并将其标记为已访问过，接着访问vi的所有未被访问过的邻接点vi1,vi2, …, vi t，并均标记已访问过，然后再按照vi1,vi2, …, vi t的次序，访问每一个顶点的所有未被访问过的邻接点，并均标记为已访问过，依次类推，直到图中所有和初始点vi有路径相通的顶点都被访问过为止。